Interview Documentation

Release v1.0.0

Aditya Raman

2022-09-11

TABLE OF CONTENTS


DATA
STRUCTURES
AND
ALGORITHMS
—
Aditya Raman
def ds_algo():
while not optimal:
time -= 1
memory -= 1
return optimal

Part I

Data Structures

CHAPTER

ONE

ARRAY

1.1 Static Array

Important: A static array is a fixed length container containing n elements indexable from the range [0 , n-1]

1.2 Complexity

	Static Array	Dynamic Array
Access	O(1)	O(1)
Search	O(n)	O(n)
Insertion	NA	O(n)
Appending	NA	O(1)
Deletion	NA	O(n)

1.3 Array Operations

class array_operation

Bases: object

All the operations associated with list

delete_at_end()

delete_at_index(*index*)

delete_ele(*ele*)

display()

get(*index*)

insert_at_end(*ele*)

insert_at_index(*ele*, *index*)

search(*key*)

```
class array_operation:
"""All the operations associated with Array"""

    def __init__(self):
        self.array = []
```

(continues on next page)

(continued from previous page)

```
    def get(self, index):
        if index < len(self.array):
            return self.array[index]
        return -1

    def insert_at_end(self, ele):
        self.array.append(ele)
        return

    def insert_at_index(self, ele, index):
        self.array.insert(index, ele)
        return

    def delete_at_end(self):
        if len(self.array) > 0:
            return self.array.pop()
        return -1

    def delete_ele(self, ele):
        res = self.search(ele)
        if res == -1:
            return res
        self.array.remove(ele)
        return res

    def delete_at_index(self, index):
        if index < len(self.array):
            return self.array.pop(index)
        return -1

    def search(self, key):
        for i in range(len(self.array)):
            if self.array[i] == key:
                return i
        return -1

    def display(self):
        for i in range(len(self.array)):
            print("{0} ".format(self.array[i]), end=" ")
        print()
```

1.4 Important Problems

CHAPTER

TWO

STACK

Important: Stack is a data structure where we store data with the rule **Last In First Out (LIFO)**.

- Used in recursion.
- Valid Parenthesis.

Warning: In python stack is implemented using list, the stack class here is just to tell about the implementation.

2.1 Complexity

	Time
Insertion	O(1)
Deletion	O(1)

2.2 Stack Operations

class Stack(*size*)

Bases: `object`

In python, stack doesn't make any sense as list has all inbuilt methods for the same But algorithm works as demonstrated below

display()

get_top()

overflow()

pop() → int

push(*ele*) → None

underflow() → bool

```
class Stack:
    def __init__(self, size):
        self.stack = []
        self.size = size  # fixed sized stack
        self.top = -1  # index of stack

    def push(self, ele) -> None:
```

(continues on next page)

(continued from previous page)

```
        if not self.overflow():
            self.top += 1
            self.stack.append(ele)
        else:
            print("Stack Overflow")

    def pop(self) -> int:
        if not self.underflow():
            self.top -= 1
            return self.stack.pop()
        print("Stack Underflow")
        return -1

    def underflow(self) -> bool:
        if self.top == -1:
            return True
        return False

    def overflow(self):
        if self.top == self.size - 1:
            return True
        return False
```

2.3 Important Problems

CHAPTER

THREE

LINKED LIST

Important: A linked-list is a sequential list of nodes that hold data which point to other nodes also containing data

3.1 Where are they used

- Used in many list, queue & stack implementation
- For creating Circular List
- Used in separate chaining, which is present certain hashtable implementations to deal with hashing collisions
- Often used in the implementation of the adjacency lists for graph

3.2 Pros and Cons

	Pros	Cons
Singly Linked List	• Uses Less Memory • Simpler Implementation	Can't easily access previous elements
Doubly Linked List	Can be Traversed backwards	Takes 2x memory

3.3 Complexity

	Singly Linked List	Doubly Linked List
Search	O(n)	O(n)
Insert at Head	O(1)	O(1)
Insert at Tail	O(1)	O(1)
Remove at Head	O(1)	O(1)
Remove at Tail	O(n)	O(1)
Remove in middle	O(n)	O(n)

class LinkedList

Bases: `object`

Operations associated with the Linked List

add_at_index (*index: int*, *val: int*) → None
Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater than the length, the node will not be inserted.

delete_at_index (*index: int*)
Delete the index-th node in the linked list, if the index is valid.

get (*index: int*)
Get the value of the index-th node in the linked list. If the index is invalid, return -1.

insert_at_head (*data*)
Insert at head

insert_at_tail (*data*)
Insert at the tail

search (*data*)
To search the given data in the Linked list and find it's first occurrence, if it is not present return -1

3.4 Node

3.4.1 Defining the node for a Linked List

Node (*data: int*)

```
def Node(data):
    data = data
    next = None
```

```
head = None
```

3.5 Search

search (*data: int*)

```
def search(data):
    """
    To search the given data in the Linked list and find it's first occurrence,
    if it is not present return -1
    """
    curr = head
    index = 0
    while curr:
        if curr.data == data:
            return index
        index += 1
        curr = curr.next
    return -1
```

3.6 Insert

3.6.1 Insert at head

insert_at_head(*data: int*)

```
def insert_at_head(data):
    """Insert at head """
    new_node = Node(data)
    new_node.next = head
    head = new_node
```

3.6.2 Insert at Tail

insert_at_tail(*data: int*)

```
def insert_at_tail(data):
    """Insert at the tail"""
    new_node = Node(data)
    # If there is no element in the linked list then add it to the head
    if head is None:
        head = new_node
    else:
        curr = head
        while curr.next:
            curr = curr.next
        curr.next = new_node
```

3.6.3 Insert at an Index

insert_at_index(*data: int*)

```
def insert_at_index(data):
    """
    Add a node of value val before the index-th node in the linked list. If index␣
↪equals to the length of linked
    list, the node will be appended to the end of linked list. If index is greater␣
↪than the length, the node will
    not be inserted.
    """
    index -= 1
    new_node = Node(val)
    if not head:
        head = new_node
        return
    curr = head
    if index < 1:
        new_node.next = head
        head = new_node
        return
    else:
        count = 0
        prev = head
        while curr:
            count += 1
            if count == index:
                new_node.next = curr.next
                curr.next = new_node
```

(continues on next page)

(continued from previous page)

```
                return
            prev = curr
            curr = curr.next
        if count == index:
            curr.next = new_node
            return
        else:
            prev.next = new_node
            return
```

3.7 Delete

3.7.1 Delete at head

delete_head()

```
def delete_head():
    if not head:
        return -1
    else:
        value = head.data
        temp = head.next
        head = None
        head = temp
        return value
```

3.7.2 Delete at Tail

delete_tail()

```
def delete_tail():
    if not head:
        return -1
    else:
        curr = head
        if not curr.next:
            value = curr.data
            head = None
            return value
        while curr.next.next:
            curr = curr.next
        value = curr.next.data
        curr.next = None
        return value
```

3.7.3 Delete at Index

delete_at_index(*index: int*)

```
def delete_at_index(index: int):
    """
    Delete the index-th node in the linked list, if the index is valid.
    """
    index -= 1
    if head is None:
        return -1
    curr = head
    if index == 0:
        value = curr.data
        head = curr.next
        return value
    elif index < 0:
        return -1
    else:
        for i in range(index - 1):
            curr = curr.next
            if curr is None:
                break
        if curr is None:
            return -1
        if curr.next is None:
            return -1
    value = curr.data
    next = curr.next.next
    curr.next = None
    curr.next = next
    return value
```

3.8 Important Problems

3.8.1 Leetcode Problems

Table 1: Linked List Cycle

Sl No	Level	Questions	Solutions	Tags
141	Easy	Linked List Cycle[1]	Python[2]	Two Pointers
142	Medium	Linked List Cycle II[3]	Python[4]	Hash Table
2	Medium	Add Two Numbers[5]	Python[6]	Traversal
19	Medium	Remove Nth Node From End of List[7]	Python[8]	Two Pointers

[1] https://leetcode.com/problems/linked-list-cycle/
[2] https://github.com/ramanaditya/data-structure-and-algorithms/tree/main/leetcode/linked-list/linked-list-cycle.py
[3] https://leetcode.com/problems/linked-list-cycle-ii/
[4] https://github.com/ramanaditya/data-structure-and-algorithms/tree/main/leetcode/linked-list/linked-list-cycle-ii.py
[5] https://leetcode.com/problems/add-two-numbers/
[6] https://github.com/ramanaditya/data-structure-and-algorithms/tree/main/leetcode/linked-list/add-two-numbers.py
[7] https://leetcode.com/problems/remove-nth-node-from-end-of-list/
[8] https://github.com/ramanaditya/data-structure-and-algorithms/tree/main/leetcode/linked-list/remove-nth-node-from-end-of-list/

Part II

Algorithms

CHAPTER

FOUR

ARRAY

4.1 Boyer–Moore majority vote algorithm

boyer_moore_voting_algorithm(*arr: list*) → int

Parameters **arr** – list

Returns int

4.1.1 Description

To find the majority element from the sequence, majority means the element should be present more than n/2 times the total length, n, of the sequence.

If no such element occurs, then algorithm can return any arbitrary element, and is not guaranteed that this element will be the mode or occurred maximum number of times.

Important:

- Linear TIme
- Constant Space

See also:

Reference : wiki[9]

4.1.2 Python

```
def boyer_moore_voting_algorithm(arr: list) -> int:
    """
    :param arr: list
    :return: int
    """
    res = arr[0]  # Initialization
    counter = 0  # Counter

    for i in range(len(arr)):
        if counter == 0:
            res = arr[i]
            counter = 1
        elif res == arr[i]:
            counter += 1
        else:
```

(continues on next page)

[9] https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_majority_vote_algorithm

(continued from previous page)

```
            counter -= 1

    return res
```

4.1.3 Implementation

Table 1: LeetCode

Sl No	Level	Questions	Solutions	Tags
169	Easy	Majority Element[10]	Python[11]	

[10] https://leetcode.com/problems/majority-element/
[11] https://github.com/ramanaditya/data-structure-and-algorithms/blob/main/leetcode/array/majority-element.py

CHAPTER

FIVE

LINKED LIST

5.1 Floyd's Tortoise and Hare

class FloydTortoiseAndHare
Implementation of Floyd's Tortoise and Hare Algorithm

check_cycle(*head*)
Return True if cycle is present else False :param head: :return:

cycle_node(*head*)
Finding the node where cycle exists :param head: :return:

class FormLinkedList(*array*, *ind*)
Class to form linked-list with cycle

createll()
Function to create linked-list with cycle :return:

class Node(*data=0*, *next=None*)
Node for the linked-list

5.1.1 Description

Floyd's cycle-finding algorithm is a pointer algorithm that uses only two pointers, which move through the sequence at different speeds. It is also called the **tortoise and the hare algorithm**

Checking the existence of the cycle in the linked-list. We can also find the node with which linked-list is linked

---

Important:

- Linear TIme
- Constant Space

See also:

Reference : wiki[12]

[12] https://en.wikipedia.org/wiki/Cycle_detection

5.1.2 Python

Check For Cycle

```
def check_cycle(self, head):
    """
    Return True is cycle is present else False
    :param head:
    :return:
    """
    # Two pointers
    tortoise, hare = head, head

    # Base Case
    while hare and hare.next:
        tortoise = tortoise.next
        hare = hare.next.next

        # Condition for cycle
        if tortoise == hare:
            return True

    # Condition when there is no cycle
    return False
```

Get the Node where cycle exists

```
def cycle_node(self, head):
    """
    Finding the node where cycle exists
    :param head:
    :return:
    """
    # Two pointers
    tortoise, hare = head, head

    while True:
        # Condition when pointer reaches to end
        if not hare or not hare.next:
            return None

        tortoise = tortoise.next
        hare = hare.next.next

        if tortoise == hare:
            break

    # Iterating over the ll to find
    tortoise = head
    while tortoise != hare:
        tortoise = tortoise.next
        hare = hare.next

    # Returning node where cycle was found
    return tortoise
```

5.1.3 Implementation

Table 1: LeetCode

Sl No	Level	Questions	Solutions	Tags
141	Easy	Linked List Cycle[13]	Python[14]	Two Pointers
142	Medium	Linked List Cycle II[15]	Python[16]	Hash-Table, Two Pointers

[13] https://leetcode.com/problems/linked-list-cycle/
[14] https://github.com/ramanaditya/data-structure-and-algorithms/blob/main/leetcode/array/majority-element.py
[15] https://leetcode.com/problems/linked-list-cycle-ii/
[16] https://github.com/ramanaditya/data-structure-and-algorithms/tree/main/leetcode/linked-list/linked-list-cycle-ii.py

CHAPTER

SIX

SEARCHING

Table 1: Search Algorithms

Sl No.	Algorithm	Worst Time	Average Time	Best Time	Memory
1	Linear Search	1	n	1	1
2	Binary Search	$log(n)$	$log(n)$	1	1

6.1 Linear Search

class LinearSearch

Bases: object

linear_search(*array*, *key*)

6.1.1 Description

This is a search algorithm that iterates over each element to find the key element

If key is found it will return the position else -1

Important:

TIme Complexity

- Worst Case: O(n)
- Average Case: O(n)
- Best Case: O(1)

Space Complexity: O(1)

6.1.2 Python

```
class LinearSearch:
    def linear_search(self, array, key):
        for i in range(len(array)):
            if array[i] == key:
                return i
        return -1
```

6.1.3 Implementation

6.2 Binary Search

class BinarySearch
Bases: object

binary_search(*array*, *key*)

6.2.1 Description

Binary Search is a sorting algorithm, applied on sorted array in which we select the mid element and compare key to the mid element if key is smaller then we search before mid else after mid.

If key is found we return the index of key else -1.

Attention:

Finding problems associated with Binary Search

- list/array is sorted
- Have to find element within the sorted list
- Can be used in case of binary values as well eg., [0,0,0,1,1,1,1]

Important:

TIme Complexity

- Worst Case: O(log(n))
- Average Case: O(log(n))
- Best Case: O(1)

Space Complexity: O(1)

6.2.2 Python

```
class BinarySearch:
    def binary_search(self, array, key):
        low = 0
        high = len(array) - 1
        while low <= high:
            mid = low + (high - low) // 2
            if array[mid] == key:
                return mid
            elif array[mid] < key:
                low = mid + 1
            else:
                high = mid - 1
        return -1
```

6.2.3 Implementation

Table 2: LeetCode

Sl No	Level	Questions	Solutions	Tags
704	Easy	Binary Search[17]	Python[18]	
367	Easy	Valid Perfect Square[19]	Python[20]	
278	Easy	First Bad Version[21]	Python[22]	
74	Medium	Search a 2D Matrix[23]	Python[24]	

[17] https://leetcode.com/problems/binary-search/
[18] https://github.com/ramanaditya/data-structure-and-algorithms/blob/main/leetcode/binary-search/binary-search.py
[19] https://leetcode.com/problems/valid-perfect-square/
[20] https://github.com/ramanaditya/data-structure-and-algorithms/blob/main/leetcode/binary-search/valid-perfect-square.py
[21] https://leetcode.com/problems/first-bad-version/
[22] https://github.com/ramanaditya/data-structure-and-algorithms/blob/main/leetcode/binary-search/first-bad-version.py
[23] https://leetcode.com/problems/search-a-2d-matrix/
[24] https://github.com/ramanaditya/data-structure-and-algorithms/blob/main/leetcode/binary-search/search-a-2d-matrix.py

CHAPTER

SEVEN

SORTING

Table 1: Sorting Algorithms

Sl No.	Algorithm	Worst Time	Average Time	Best Time	Memory	Stability
1	Bubble Sort	n^2	n^2	n	1	Stable
2	Selection Sort	n^2	n^2	n^2	1	Unstable
3	Insertion Sort	n^2	n^2	n^2	1	Stable
4	Merge Sort	$n * log(n)$	$n * log(n)$	$n * log(n)$	n	Stable
5	Quick Sort	$n * log(n)$	$n * log(n)$	n^2	$n * log(n)$	Unstable

Note:

- **Stable** : Relative position of equal elements after sorting remains same.
- **In-place Sorting** : Sorting Input elements without having backup, thus unsorted form is lost.

7.1 Bubble Sort

class bubbleSort
Bases: object

bubble_sort(*data*)

7.1.1 Description

Important:

TIme Complexity

- Worst Case: n^2
- Average Case: n^2
- Best Case: n

Space Complexity: $O(1)$
In Place Sorting
Stable Sorting

7.1.2 Python

```
def bubble_sort(data):
    for i in range(len(data) - 1):
        for j in range(0, len(data) - i - 1):
            if data[j] > data[j + 1]:
                data[j], data[j + 1] = data[j + 1], data[j]
    return data
```

7.1.3 Implementation

7.2 Selection Sort

class selectionSort
Bases: object

selection_sort(*data*)

7.2.1 Description

Important:

TIme Complexity

- Worst Case: n^2
- Average Case: n^2
- Best Case: n^2

Space Complexity: $O(1)$
In Place Sorting
Unstable Sorting

7.2.2 Python

```
def selection_sort(data):
    for i in range(len(data)):
        min_index = i
        for j in range(i + 1, len(data)):
            if data[min_index] > data[j]:
                min_index = j
        data[i], data[min_index] = data[min_index], data[i]
    return data
```

7.2.3 Implementation

7.3 Insertion Sort

class insertionSort
Bases: object

insertion_sort(*data*)

7.3.1 Description

Tip:

- Insertion sort is used when number of elements is small.
- It can also be useful when input array is almost sorted, only few elements are misplaced in complete big array.

Important:

TIme Complexity

- Worst Case: n^2
- Average Case: n^2

- Best Case: n^2

Space Complexity: $O(1)$
In Place Sorting
Stable Sorting

7.3.2 Python

```
def insertion_sort(data):
    for i in range(1, len(data)):
        key = data[i]
        j = i - 1
        while j >= 0 and key < data[j]:
            data[j + 1] = data[j]
            j -= 1
        data[j + 1] = key
    return data
```

7.3.3 Implementation

7.4 Merge Sort

class MergeSort
Bases: object

merge_sort(*data*)

7.4.1 Description

Important:

TIme Complexity

- Worst Case: $n * log(n)$
- Average Case: $n * log(n)$
- Best Case: $n * log(n)$

Space Complexity: $O(n)$
In Place Sorting
Stable Sorting

7.4.2 Python

```
def merge_sort(data):
    if len(data) > 1:
        mid = len(data) // 2
        left = data[:mid]
        right = data[mid:]

        merge_sort(left)
        merge_sort(right)

        i = 0
        j = 0
        k = 0

        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                data[k] = left[i]
                i += 1
            else:
                data[k] = right[j]
                j += 1
            k += 1

        while i < len(left):
            data[k] = left[i]
            i += 1
            k += 1

        while j < len(right):
            data[k] = right[j]
            j += 1
            k += 1
```

7.4.3 Implementation

7.5 Quick Sort

class QuickSort

Bases: object

partition(*data*, *low*, *high*)

quick_sort(*data*, *low*, *high*)

7.5.1 Description

Important:

TIme Complexity

- Worst Case: $n * log(n)$
- Average Case: $n * log(n)$
- Best Case: n^2

Space Complexity: $O(n * log(n))$

In Place Sorting
Unstable Sorting

7.5.2 Python

Algorithm for partition

```
def partition(data, low, high):
    pivot = data[high]
    i = low  # To keep the index of element smaller than pivot
    j = low  # To keep the index of element greater than pivot

    while j < high:
        if data[j] < pivot:
            data[j], data[i] = data[i], data[j]
            i += 1
        j += 1

    data[i], data[high] = data[high], data[i]

    return i
```

Recursive Algorithm for quicksort

```
def quick_sort(data, low, high):
    if low < high:
        pivot = partition(data, low, high)
        quick_sort(data, low, pivot - 1)
        quick_sort(data, pivot + 1, high)
```

7.5.3 Implementation

CHAPTER

EIGHT

TWO POINTERS

8.1 Description

8.1.1 Variations

- Same Direction
- Opposite Direction

8.1.2 Implementation

Table 1: LeetCode

Sl No	Level	Questions	Solutions	Tags
26	Easy	Remove Duplicates from Sorted Array[25]	Python[26]	Two Pointers
189	Easy	Rotate Array[27]	Python[28]	Two Pointers
167	Easy	Two Sum II - Input array is sorted[29]	Python[30]	Two Pointers
11	Medium	Container With Most Water[31]	Python[32]	Two Pointers
238	Medium	Product of Array Except Self[33]	Python[34]	Two Pointerss

[25] https://leetcode.com/problems/remove-element/
[26] https://github.com/ramanaditya/data-structure-and-algorithms/blob/main/leetcode/array/remove-duplicates-from-sorted-array.py
[27] https://leetcode.com/problems/rotate-array/
[28] https://github.com/ramanaditya/data-structure-and-algorithms/blob/main/leetcode/array/rotate-array.py
[29] https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/
[30] https://github.com/ramanaditya/data-structure-and-algorithms/blob/main/leetcode/array/two-sum-ii-input-array-is-sorted.py
[31] https://leetcode.com/problems/container-with-most-water/
[32] https://github.com/ramanaditya/data-structure-and-algorithms/blob/main/leetcode/array/container-with-most-water.py
[33] https://leetcode.com/problems/product-of-array-except-self/
[34] https://github.com/ramanaditya/data-structure-and-algorithms/blob/main/leetcode/array/product-of-array-except-self.py

Part III

Networking

CHAPTER

NINE

IP ADDRESS

- IP Address is assigned to an **Interface** port

Important: 8 bits = 1 Byte = 1 Octet

9.1 IPv4

- 32 bits (4 byte) address format.

Part IV

Interview

CHAPTER

TEN

INTERVIEW PREPARATION

10.1 Interview Best Practices

- **Be Authentic**
  - Don't lie but this does not mean you have to tell everything.
  - Be Strategic and concise.
- **Sell Yourself**
  - Focus on preparing authentic answers that highlight your greatest strengths.
  - Tell good, authentic, relevant stories about your experience.
- **Be Concise**
  - Don't bore interviewers with the long answers.
  - Don't go beyond 2 minutes for a single answer.
  - If you bore them, you lose them.
  - Emphasise your most impressive points.
- **Show Enthusiasm**
  - Demonstrate convincing enthusiasm for the company and the position.
  - Ask Questions
  - Be Motivated

10.2 Common Mistakes

- **Lack of Professionalism**
  - Never be late for an interview
  - How you Dress matters.
  - How you present yourself.
  - Job Etiquette like saying Thank You.
- **Lack of Preparation**
  - Analyzing Job Interviews
  - Thinking to key questions before Job Interview
  - Good Behavioural Stories
- **Lack of Content**
  - Lame descriptions

    - Lcck of Preparation
    - Don't give generic answers.
- **Negativity**

10.3 Types of Inerviews

10.3.1 Phone Interview

- The one to one in Person Interview
- The Video Interview
- The Panel Interview
- The Group Interview

10.3.2 Types of Interviewers

- External Recruiter: Doesn't work for the company but can assign people to the company and get paid.
- Internal Recruiter or HR Rep: Part of the Company and generally check for the authenticity and personality.
- The Hiring Manager: Can dig deeper into the technical Questions and even for the personality.
- Senior Level Management: More likely to see the work or hands-on experience and will check for the potential people.
- Direct Report: You might be interviewed with someone who will work for you.

10.3.3 You need to do

- Building Report
- Showing Respect for the current Team
- Show respect for the way they do things now.
- Hope for the best.

10.4 Analyzing the Job Description

Note: A written description of the qualifications, duties and responsibilities of a position.

- Work your network to find out more about the position.
- Try searching the job description for the similar roles at similar companies.
- Customizing your approach is the key to standing out from the pack.

10.4.1 To create your Job Description

- Identify Competencies: Highlight the description of the position multiple times in form of your work experience.
- Identify Themes: Most important of your work should be at the top which will also include the demand of the company.
- Identify your Selling Points: Emphasize your strengths and be prepared for the weaknesses.
- Identify Gaps or Issues: be Honest, Identify your weakness, what could be perceived as weaknesses by others.
- Anticipate Question

10.5 Answering Inappropriate Questions

Tip:

- Decline or deflect the questions that include the following topics
- Gender, Age, Marital Status, Where you are from, originally.

Where you are from originally? I think I am <Your current location>, since I residing here for a long time.

I don't think that's too relevant, but I would love to tell you more about my programming experience.

10.6 General Standards

- Dress a little bit more formally, than the company's dress code.
- Hair and nails should be clean and groomed
- Your dress should be clean and wrinkles free.
- Wear knee lengths skirts.
- If you are not asked to wear suit, avoid it.
- At last make it your choice.

10.6.1 Why dressing is important

- To reduce distractions
- Not to let people judge you based upon your dress.
- They could listen more about your skills.

10.7 Overcoming Nerves

- **Do your homework**
  - Practice more and keep a not of your points
  - Don't repeat the same words time and again
- **Accentuate the Positive**
  - Fake Smile can create positive impact but only if you can smile internally.
  - Don't think more or out of context during the interview process.
- **Psyche yourself up**

CHAPTER

ELEVEN

NETWORKING

Important: Networking is interacting with others to exchange information and develop professional or social contacts.

- Seeking support from people who already know and respect you
- **Define your network**
    1. Name
    2. How you are connected
    3. Potential
    4. Reasons
    5. Contacts
- **Expand your thinking**
- **Reach Out**
    - **Strong Connections: Consider these details**
        - Clarify what you are looking for
        - Build Credibility
        - Ask open ended questions
        - Ask for Introductions
        - Be appreciative
        - Note next steps
    - Medium Connections
    - Low Connections
    - Targeting Specific Employers: Reach out to the employees of your target companies.
    - How to Contact: Don't spam out for messages just for networking favour.
    - Following Up: Thank them for even their smallest help. Thank them when you get a job.
- **LinkedIn**
    - Setting up profile: put clear, professional photo.
    - About Section: Write a nice summary without using generic words like experienced, learning etc. Update it regularly.
    - **How to be found:**
        - Change the public url to your name or username.
        - Adapt Public Visibility

        * Use Keywords
        * Complete your Contact Details
        * List only relevant Skills
    - **LinkedIn Contacts:**
        * Your Contacts
        * Don't Spam
        * Informational Interviews
    - Finding Jobs
    - Jumping Gatekeepers
- **Keep a clean Web Presence**
    - Don't use bad spelling or grammars.
    - Personal Website
    - **GitHub**
        * Fill out your profile
        * Be mindful about pinned repository
        * Cleaned up your starred repository
        * Include an informative readme

Part V

Technical Interview

CHAPTER

TWELVE

PYTHON INTERVIEW QUESTIONS

Memory Management in Python Python is a high-level programming language that's implemented in the C programming language. The Python memory manager manages Python's memory allocations. There's a private heap that contains all Python objects and data structures. The Python memory manager manages the Python heap on demand. The Python memory manager has object-specific allocators to allocate memory distinctly for specific objects such as int, string, etc… Below that, the raw memory allocator interacts with the memory manager of the operating system to ensure that there's space on the private heap. The Python memory manager manages chunks of memory called "Blocks". A collection of blocks of the same size makes up the "Pool". Pools are created on Arenas, chunks of 256kB memory allocated on heap=64 pools. If the objects get destroyed, the memory manager fills this space with a new object of the same size. Methods and variables are created in Stack memory. A stack frame is created whenever methods and variables are created. These frames are destroyed automatically whenever methods are returned. Objects and instance variables are created in Heap memory. As soon as the variables and functions are returned, dead objects will be garbage collected. It is important to note that the Python memory manager doesn't necessarily release the memory back to the Operating System, instead memory is returned back to the python interpreter. Python has a small objects allocator that keeps memory allocated for further use. In long-running processes, you may have an incremental reserve of unused memory.

Part VI

Resources

CHAPTER

THIRTEEN

RESOURCES

13.1 Books

- CLRS[35]
- Cracking the Coding Interview[36]
- Elements of Programming Interviews[37]

13.2 Competitive Coding

- Codechef[38]
- Codeforces[39]
- HackerEarth[40]
- HackerRank[41]
- Leetcode[42]

13.3 Online Study Materials

- Free Code Camp[43]
- GeeksforGeeks[44]

[35] https://mitpress.mit.edu/books/introduction-algorithms-third-edition
[36] http://www.crackingthecodinginterview.com/
[37] https://elementsofprogramminginterviews.com/
[38] https://www.codechef.com/
[39] https://codeforces.com/
[40] https://www.hackerearth.com/challenges/
[41] https://www.hackerrank.com/
[42] https://leetcode.com/
[43] https://www.freecodecamp.org/
[44] https://www.geeksforgeeks.org/

CHAPTER

FOURTEEN

CREDITS

We recognise and thank everyone who contributed or whose code or related documentation or hyperlink has been used in the documentation.

There might be cases when your name might not be in the list please feel free to raise a PR to the repository along with the details of your work.

Table 1: We thank You

Sl. No.	Name	Contact
1	Pamela Skillings	
2	William Fiset	

CHAPTER

FIFTEEN

COLLABORATORS

We really thank each member of our team, who have made this documentation really helpful for the people. For their unique contribution we are mentioning them on this page.

Table 1: We thank You

Sl. No.	Name	Contact
1	Aditya Raman[45]	LinkedIn[46], GitHub[47]

[45] https://www.ramanaditya.com
[46] https://www.linkedin.com/in/ramanaditya/
[47] https://github.com/ramanaditya

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

U