**ANALYSIS OF ALGORITHMS AND COMPUTATIONAL COMPLEXITY**

**CS 5329   
SPRING 2016**

**ANUSHA VATTAM  
05/05/2016**

****

**SUBMITTED TO Dr.C.J.HWANG   
DEPARTMENT OF COMPUTER SCIENCE  
TEXAS STATE UNIVERSITY**

**TABLE OF CONTENTS**

**Chapter 1 - Insertion and Merge Sorts 6  
 1.1** Problem Description and Explanation 6  
 **1.1.1** Insertion Sort 6  
 **1.1.2** Merge Sort 7  
 **1.2** Tabulation, Output and Analysis 8  
 **1.2.1** Input arrays, sorted array and analysis 8  
 **1.2.1.1** Insertion Sort 8  
 **1.2.1.2** Merge Sort 12  
 **1.2.2** With more values 14  
 **1.2.2.1** Insertion Sort 14  
 **1.2.2.2** Merge Sort 30  
 **1.2.3** Output and Graphical analysis of Algorithm 39  
 **1.2.3.1** Insertion Sort 39  
 **1.2.3.2** Merge Sort 40  
 **1.3** Source Programs 42  
 **1.3.1** Insertion Sort 42  
 **1.3.2** Merge Sort 44

**Chapter 2 - Maximum Sub-Array problem 48  
 2.1** Problem Description and Explanation 48  
 **2.2** Tabulation, Output and Analysis 50  
 **2.2.1** Input arrays, sorted array and analysis 50  
 **2.2.2** With more values 57  
 **2.2.3** Output and Graphical analysis of Algorithm 72  
 **2.3** Source Programs 73

**Chapter 3 - Quick Sort or Randomized Quick Sort 78  
 3.1** Problem Description and Explanation 78  
 **3.2** Tabulation, Output and Analysis 79   
 **3.2.1** Input arrays, sorted array and analysis 81   
 **3.2.1.1** Quick Sort 81  
 **3.2.1.2** Randomized Quick Sort 87  
 **3.2.2** With more values 91  
 **3.2.2.1** Quick Sort 91  
 **3.2.2.2** Randomized Quick Sort 105  
 **3.2.3** Output and Graphical analysis of Algorithm 115  
 **3.2.3.1** Quick Sort 115  
  **3.2.3.2** Randomized Quick Sort 115  
 **3.3** Source Programs 116

**Chapter 4 - Heap Sort 120  
 4.1** Problem Description and Explanation 120  
 **4.2** Tabulation, Output and Analysis 122  
 **4.2.1** Input arrays, sorted array and analysis 122  
 **4.2.2** With more values 126  
 **4.2.3** Output and Graphical analysis of Algorithm 140  
 **4.2.4** Graphical comparisons of Merge, Heap and Quick Sorts 141  
 **4.3** Source Programs 141

**Chapter 5 - Hashing and Hash Tables 146  
 5.1** Problem Description and Explanation 146  
 **5.2** Tabulation, Output and Analysis 147  
 **5.2.1** Division Method 147  
 **5.2.2** Multiplication Method 155  
 **5.2.3** Quadratic Probing Method 163  
 **5.3** Source Programs 168  
 **5.3.1** Division Method 168  
 **5.3.2** Multiplication Method 172  
 **5.3.3** Quadratic Probing Method 176

**Chapter 6 - Huffman Codes Problem and Greedy Algorithms 180  
 6** Problem Description and Explanation 180 **6.1** Variable length encoding 181  
 **6.1.1** Input Texts for Compression 181  
 **6.1.2** Compressed 0’s and 1’s 184  
 **6.1.3** Decompressed Texts 184  
 **6.2** Fixed length encoding 185  
 **6.2.1** Input Texts for Compression 185  
 **6.2.2** Compressed 0’s and 1’s 188  
 **6.2.3** Decompressed Texts 188  
 **6.3** Comparisons 189  
 **6.4** Source Programs 194  
 **6.4.1** Variable Length Encoding 194  
 **6.4.2** Fixed Length Encoding 201

**Chapter 7 - Priority Queue Problem using Heapify or Heap Sort 208  
 7.1** Problem Description and Explanation 208  
 **7.2** Tabulation, Output and Analysis 210  
 **7.2.1** Heapified Wait Queue 214  
 **7.2.2** Result after 3 moves from Wait Queue to Ready Queue 218  
 **7.2.3** Result after removing 2 job’s from Ready Queue 223  
 **7.2.4** Result after Adding, Moving and Removing job’s 228  
 **7.2.5** Changing Priority of job in Wait Queue 233  
 **7.2.6** Doing More 236  
 **7.3** Source Programs 251

**Chapter 8 - Matrix Chain Multiplication Problem 266  
 8** Problem Description and Explanation 266 **8.1** m[i,j] and s[i,j] tabulations with same matrices as in the text book 268  
 **8.2** m[i,j] and s[i,j] tabulations of other matrices 271  
 **8.3** Source Program 289

**Chapter 9 - Binary Search Tree Problem 292  
 9.1** Problem Description and Explanation 292  
 **9.2** Tabulation, Output and Analysis 294  
 **9.2.1** Unsorted Student Records 314  
 **9.2.2** In-Order Traversal of Binary Search Tree 315  
 **9.2.3** Searching 2 node’s in Binary Search Tree 318  
 **9.2.4** Deleting 2 node’s from Binary Search Tree 319  
 **9.2.5** Inserting one node into Binary Search Tree 323  
 **9.2.6** More Set’s of Student Records 327  
 **9.3** Source Program 410

**Chapter 10 - Graph Algorithm and Breadth First Search Tree 417  
 10.1** Problem Description and Explanation 417  
 **10.2** Tabulation, Output and Analysis 419  
 **10.2.1** Graph of 10 vertices 424  
 **10.2.2** Searching 1st Vertex 425  
 **10.2.2.1** Complete Tree 425  
 **10.2.2.2** Shortest Path from root node 426  
 **10.2.3** Searching 2nd Vertex 426  
 **10.2.3.1** Complete Tree 426  
 **10.2.3.2** Shortest Path from root node 427  
 **10.2.4** Searching 3rd Vertex 428  
 **10.2.4.1** Complete Tree 428  
 **10.2.4.2** Shortest Path from root node 429  
 **10.3** Source Program 429

**Chapter 11 - Conclusion 435**

**CHAPTER 1**

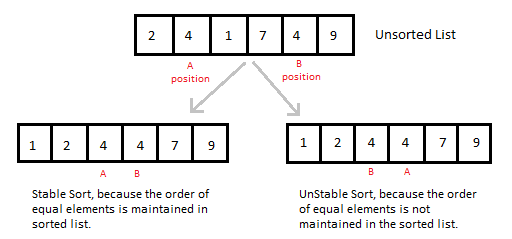
**INSERTION AND MERGE SORTS**

**1.1 Problem Description And Explanation**

**1.1.1 INSERTION SORT**

It is a simple Sorting algorithm which sorts the array by shifting elements one by one. Following are some of the important characteristics of Insertion Sort.

1. It has one of the simplest implementation
2. It is efficient for smaller data sets, but very inefficient for larger lists.
3. Insertion Sort is adaptive, that means it reduces its total number of steps if given a partially sorted list, hence it increases its efficiency.
4. It is better than Selection Sort and Bubble Sort algorithms.
5. Its space complexity is less, like Bubble Sorting, inerstion sort also requires a single additional memory space.
6. It is **Stable**, as it does not change the relative order of elements with equal keys



#### How Insertion Sorting Works



#### Sorting using Insertion Sort Algorithm

int a[6] = {5, 1, 6, 2, 4, 3};

int i, j, key;

for(i=1; i<6; i++)

{

key = a[i];

j = i-1;

while(j>=0 && key < a[j])

{

a[j+1] = a[j];

j--;

}

a[j+1] = key;

}

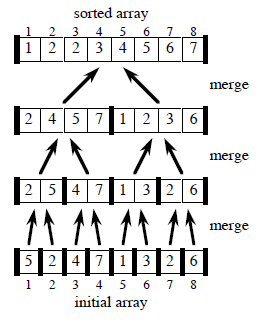
To understand the above simple insertion sort algorithm. We took an array with 6 integers. We took a variable **key**, in which we put each element of the array, in each pass, starting from the second element, that is **a[1]**.

Then using the while loop, we iterate, until **j** becomes equal to zero or we find an element which is greater than **key**, and then we insert the key at that position.

In the above array, first we pick 1 as key, we compare it with 5(element before 1), 1 is smaller than 5, we shift 1 before 5. Then we pick 6, and compare it with 5 and 1, no shifting this time. Then 2 becomes the key and is compared with, 6 and 5, and then 2 is placed after 1. And this goes on, until complete array gets sorted.

**1.1.2 MERGE SORT**

Merge Sort follows the rule of **Divide and Conquer**. But it doesn't divides the list into two halves. In merge sort the unsorted list is divided into N sublists, each having one element, because a list of one element is considered sorted. Then, it repeatedly merge these sublists, to produce new sorted sublists, and at lasts one sorted list is produced.



Merge Sort is quite fast, and has a time complexity of **O(n log n)**. It is also a stable sort, which means the "equal" elements are ordered in the same order in the sorted list.

MERGE(A, p, q, r)   
1 n1 = q - p + 1  
2 n2 = r - q  
3 let L[1. . n1 + 1] and R[1 . . n2 + 1] be new arrays  
4 for i = 1 to n1  
5 L[i] = A[p + i - 1]  
6 for j = 1 to n2  
7 R[j ] = A[q + j ]  
8 L[n1 + 1] = ∞  
9 R[n2 + 1] = ∞  
10 i = 1  
11 j = 1  
12 for k = p to r  
13 if L[i] ≤ R[j ]  
14 A[k] = L[i]  
15 i = i + 1  
16 else A[k] = R[j ]  
17 j = j + 1

The procedure MERGE-SORT(A, p, r) sorts the elements in the subarray A[p . . r]. If p ≥ r, the subarray has at most one element and is therefore already sorted. Otherwise, the divide step simply computes an index q that partitions A[p . . r] into two subarrays: A[p . . q], containing ┌ n/2 ┐ elements, and A[q + 1. . r], containing L n/2 ˩ elements.

MERGE-SORT(A, p, r)

1 if p < r

2 q = L (p + r)/2 ˩

3 MERGE-SORT(A, p, q)

4 MERGE-SORT(A, q + 1, r)

5 MERGE(A, p, q, r)

**1.2 Tabulation, Output and Analysis**

**COMBINING RESULTS OF INSERTION SORT ALGORITHM AND MERGE SORT ALGORITHM**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **Actual Count MERGE** | **O(nlogn)** | **Actual Count INSERTION** | **O(n2)** |
| 5 | 7 | 3.495 | 8 | 25 |
| 10 | 21 | 10 | 16 | 100 |
| 12 | 29 | 12.95 | 20 | 144 |
| 15 | 42 | 17.64 | 38 | 225 |
| 20 | 56 | 26.02 | 57 | 400 |
| 25 | 81 | 34.94 | 120 | 625 |
| 30 | 112 | 44.31 | 149 | 900 |
| 35 | 130 | 54.04 | 278 | 1225 |
| 40 | 146 | 64.08 | 291 | 1600 |
| 50 | 157 | 84.94 | 303 | 2500 |

**1.2.1 Input arrays, sorted array and analysis**

**1.2.1.1 INSERTION SORT**

**PROGRAM OUTPUT**

1.

Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:24:06) [MSC v.1600 32 bit (Intel)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

Enter number of values 5

Enter Value 5

Enter Value 6

Enter Value 1

Enter Value 2

Enter Value 0

Unsorted data input [5, 6, 1, 2, 0]

Sorted data output [0, 1, 2, 5, 6]

Actual Count 8

2. Enter number of values : 10

Enter Value : 9

Enter Value : 0

Enter Value : 8

Enter Value : 3

Enter Value : 2

Enter Value : 7

Enter Value : 32

Enter Value : 6

Enter Value : 23

Enter Value : 11

Unsorted data input : [9, 0, 8, 3, 2, 7, 32, 6, 23, 11]

Sorted data output : [0, 2, 3, 6, 7, 8, 9, 11, 23, 32]

Actual Count : 16

3.

Enter number of values : 12

Enter Value : 90

Enter Value : 3

Enter Value : 5

Enter Value : 2

Enter Value : 7

Enter Value : 4

Enter Value : 22

Enter Value : 17

Enter Value : 18

Enter Value : 14

Enter Value : 34

Enter Value : 65

Unsorted data input : [90, 3, 5, 2, 7, 4, 22, 17, 18, 14, 34, 65]

Sorted data output : [2, 3, 4, 5, 7, 14, 17, 18, 22, 34, 65, 90]

Actual Count : 20

4.

Enter number of values : 15

Enter Value : 12

Enter Value : 29

Enter Value : 30

Enter Value : 35

Enter Value : 49

Enter Value : 52

Enter Value : 65

Enter Value : 76

Enter Value : 80

Enter Value : 55

Enter Value : 33

Enter Value : 86

Enter Value : 43

Enter Value : 22

Enter Value : 33

Unsorted data input : [12, 29, 30, 35, 49, 52, 65, 76, 80, 55, 33, 86, 43, 22, 33]

Sorted data output : [12, 22, 29, 30, 33, 33, 35, 43, 49, 52, 55, 65, 76, 80, 86]

Actual Count : 38

5.

Enter number of values : 20

Enter Value : 5

Enter Value : 3

Enter Value : 4

Enter Value : 8

Enter Value : 9

Enter Value : 0

Enter Value : 21

Enter Value : 43

Enter Value : 54

Enter Value : 65

Enter Value : 76

Enter Value : 87

Enter Value : 9

Enter Value : 8

Enter Value : 11

Enter Value : 21

Enter Value : 13

Enter Value : 15

Enter Value : 26

Enter Value : 23

Unsorted data input : [5, 3, 4, 8, 9, 0, 21, 43, 54, 65, 76, 87, 9, 8, 11, 21, 13, 15, 26, 23]

Sorted data output : [0, 3, 4, 5, 8, 8, 9, 9, 11, 13, 15, 21, 21, 23, 26, 43, 54, 65, 76, 87]

Actual Count : 57

6.

Enter number of values : 25

Enter Value : 100

Enter Value : 200

Enter Value : 0

Enter Value : 1900

Enter Value : 300

Enter Value : 400

Enter Value : 600

Enter Value : 900

Enter Value : 2

Enter Value : 4

Enter Value : 7

Enter Value : 9

Enter Value : 2

Enter Value : 4

Enter Value : 77

Enter Value : 22

Enter Value : 65

Enter Value : 32

Enter Value : 21

Enter Value : 54

Enter Value : 87

Enter Value : 98

Enter Value : 54

Enter Value : 64

Enter Value : 87

Unsorted data input : [100, 200, 0, 1900, 300, 400, 600, 900, 2, 4, 7, 9, 2, 4, 77, 22, 65, 32, 21, 54, 87, 98, 54, 64, 87]

Sorted data output : [0, 2, 2, 4, 4, 7, 9, 21, 22, 32, 54, 54, 64, 65, 77, 87, 87, 98, 100, 200, 300, 400, 600, 900, 1900]

Actual Count : 120

7.

Enter number of values : 30

Enter Value : 5

Enter Value : 10

Enter Value : 15

Enter Value : 20

Enter Value : 2

Enter Value : 4

Enter Value : 5

Enter Value : 25

Enter Value : 50

Enter Value : 34

Enter Value : 67

Enter Value : 89

Enter Value : 32

Enter Value : 22

Enter Value : 11

Enter Value : 26

Enter Value : 90

Enter Value : 44

Enter Value : 68

Enter Value : 78

Enter Value : 23

Enter Value : 45

Enter Value : 56

Enter Value : 46

Enter Value : 43

Enter Value : 58

Enter Value : 26

Enter Value : 88

Enter Value : 77

Enter Value : 44

Unsorted data input : [5, 10, 15, 20, 2, 4, 5, 25, 50, 34, 67, 89, 32, 22, 11, 26, 90, 44, 68, 78, 23, 45, 56, 46, 43, 58, 26, 88, 77, 44]

Sorted data output : [2, 4, 5, 5, 10, 11, 15, 20, 22, 23, 25, 26, 26, 32, 34, 43, 44, 44, 45, 46, 50, 56, 58, 67, 68, 77, 78, 88, 89, 90]

Actual Count : 149

8.

Enter number of values : 35

Enter Value : 50

Enter Value : 40

Enter Value : 30

Enter Value : 25

Enter Value : 18

Enter Value : 15

Enter Value : 12

Enter Value : 10

Enter Value : 5

Enter Value : 53

Enter Value : 42

Enter Value : 33

Enter Value : 26

Enter Value : 19

Enter Value : 16

Enter Value : 13

Enter Value : 11

Enter Value : 21

Enter Value : 31

Enter Value : 41

Enter Value : 51

Enter Value : 61

Enter Value : 3

Enter Value : 25

Enter Value : 23

Enter Value : 21

Enter Value : 12

Enter Value : 18

Enter Value : 22

Enter Value : 20

Enter Value : 0

Enter Value : 12

Enter Value : 67

Enter Value : 28

Enter Value : 87

Unsorted data input : [50, 40, 30, 25, 18, 15, 12, 10, 5, 53, 42, 33, 26, 19, 16, 13, 11, 21, 31, 41, 51, 61, 3, 25, 23, 21, 12, 18, 22, 20, 0, 12, 67, 28, 87]

Sorted data output : [0, 3, 5, 10, 11, 12, 12, 12, 13, 15, 16, 18, 18, 19, 20, 21, 21, 22, 23, 25, 25, 26, 28, 30, 31, 33, 40, 41, 42, 50, 51, 53, 61, 67, 87]

Actual Count : 303

9.

Enter number of values : 40

Enter Value : 40

Enter Value : 31

Enter Value : 7

Enter Value : 9

Enter Value : 4

Enter Value : 49

Enter Value : 53

Enter Value : 32

Enter Value : 9

Enter Value : 8

Enter Value : 5

Enter Value : 3

Enter Value : 99

Enter Value : 0

Enter Value : 6

Enter Value : 40

Enter Value : 80

Enter Value : 56

Enter Value : 45

Enter Value : 47

Enter Value : 42

Enter Value : 17

Enter Value : 18

Enter Value : 19

Enter Value : 0

Enter Value : 67

Enter Value : 55

Enter Value : 433

Enter Value : 66

Enter Value : 71

Enter Value : 74

Enter Value : 78

Enter Value : 32

Enter Value : 47

Enter Value : 50

Enter Value : 63

Enter Value : 62

Enter Value : 25

Enter Value : 60

Enter Value : 43

Unsorted data input : [40, 31, 7, 9, 4, 49, 53, 32, 9, 8, 5, 3, 99, 0, 6, 40, 80, 56, 45, 47, 42, 17, 18, 19, 0, 67, 55, 433, 66, 71, 74, 78, 32, 47, 50, 63, 62, 25, 60, 43]

Sorted data output : [0, 0, 3, 4, 5, 6, 7, 8, 9, 9, 17, 18, 19, 25, 31, 32, 32, 40, 40, 42, 43, 45, 47, 47, 49, 50, 53, 55, 56, 60, 62, 63, 66, 67, 71, 74, 78, 80, 99, 433]

Actual Count : 278

10.

Enter number of values : 50

Enter Value : 2

Enter Value : 6

Enter Value : 1

Enter Value : 0

Enter Value : 67

Enter Value : 9

Enter Value : 3

Enter Value : 0

Enter Value : 4

Enter Value : 6

Enter Value : 2

Enter Value : 1

Enter Value : 8

Enter Value : 67

Enter Value : 56

Enter Value : 45

Enter Value : 34

Enter Value : 23

Enter Value : 12

Enter Value : 23

Enter Value : 16

Enter Value : 4

Enter Value : 7

Enter Value : 3

Enter Value : 8

Enter Value : 2

Enter Value : 5

Enter Value : 45

Enter Value : 63

Enter Value : 69

Enter Value : 58

Enter Value : 52

Enter Value : 47

Enter Value : 57

Enter Value : 54

Enter Value : 55

Enter Value : 00

Enter Value : 90

Enter Value : 98

Enter Value : 96

Enter Value : 87

Enter Value : 84

Enter Value : 83

Enter Value : 97

Enter Value : 05

Enter Value : 77

Enter Value : 99

Enter Value : 66

Enter Value : 567

Enter Value : 890

Unsorted data input : [2, 6, 1, 0, 67, 9, 3, 0, 4, 6, 2, 1, 8, 67, 56, 45, 34, 23, 12, 23, 16, 4, 7, 3, 8, 2, 5, 45, 63, 69, 58, 52, 47, 57, 54, 55, 0, 90, 98, 96, 87, 84, 83, 97, 5, 77, 99, 66, 567, 890]

Sorted data output : [0, 0, 0, 1, 1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 8, 8, 9, 12, 16, 23, 23, 34, 45, 45, 47, 52, 54, 55, 56, 57, 58, 63, 66, 67, 67, 69, 77, 83, 84, 87, 90, 96, 97, 98, 99, 567, 890]

Actual Count : 291

**1.2.1.2 MERGE SORT**

1.

Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:24:06) [MSC v.1600 32 bit (Intel)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

Enter number of values 5

Enter Value 9

Enter Value 0

Enter Value 8

Enter Value 5

Enter Value 3

Unsorted data input [9, 0, 8, 5, 3]

Merge Sort sorted output: [0, 3, 5, 8, 9]

Actual Count : 7

2.

Enter number of values 10

Enter Value 3

Enter Value 6

Enter Value 9

Enter Value 2

Enter Value 1

Enter Value 0

Enter Value 4

Enter Value 8

Enter Value 23

Enter Value 45

Unsorted data input [3, 6, 9, 2, 1, 0, 4, 8, 23, 45]

Merge Sort sorted output: [0, 1, 2, 3, 4, 6, 8, 9, 23, 45]

Actual Count : 21

3.

Enter number of values 12

Enter Value 3

Enter Value 5

Enter Value 2

Enter Value 1

Enter Value 8

Enter Value 5

Enter Value 2

Enter Value 4

Enter Value 1

Enter Value 0

Enter Value 22

Enter Value 32

Unsorted data input [3, 5, 2, 1, 8, 5, 2, 4, 1, 0, 22, 32]

Merge Sort sorted output: [0, 1, 1, 2, 2, 3, 4, 5, 5, 8, 22, 32]

Actual Count : 29

4.

Enter number of values 15

Enter Value 22

Enter Value 33

Enter Value 11

Enter Value 55

Enter Value 66

Enter Value 3

Enter Value 21

Enter Value 43

Enter Value 56

Enter Value 32

Enter Value 12

Enter Value 14

Enter Value 16

Enter Value 18

Enter Value 10

Unsorted data input [22, 33, 11, 55, 66, 3, 21, 43, 56, 32, 12, 14, 16, 18, 10]

Merge Sort sorted output: [3, 10, 11, 12, 14, 16, 18, 21, 22, 32, 33, 43, 55, 56, 66]

Actual Count : 42

5.

Enter number of values 20

Enter Value 12

Enter Value 15

Enter Value 11

Enter Value 10

Enter Value 28

Enter Value 26

Enter Value 32

Enter Value 21

Enter Value 43

Enter Value 15

Enter Value 19

Enter Value 14

Enter Value 10

Enter Value 14

Enter Value 43

Enter Value 54

Enter Value 65

Enter Value 76

Enter Value 87

Enter Value 98

Unsorted data input [12, 15, 11, 10, 28, 26, 32, 21, 43, 15, 19, 14, 10, 14, 43, 54, 65, 76, 87, 98]

Merge Sort sorted output: [10, 10, 11, 12, 14, 14, 15, 15, 19, 21, 26, 28, 32, 43, 43, 54, 65, 76, 87, 98]

Actual Count : 56

6.

Enter number of values 25

Enter Value 0

Enter Value 9

Enter Value 8

Enter Value 5

Enter Value 8

Enter Value 5

Enter Value 4

Enter Value 3

Enter Value 9

Enter Value 21

Enter Value 78

Enter Value 95

Enter Value 62

Enter Value 64

Enter Value 57

Enter Value 58

Enter Value 594

Enter Value 562

Enter Value 55

Enter Value 58

Enter Value 65

Enter Value 574

Enter Value 582

Enter Value 52

Enter Value 56

Unsorted data input [0, 9, 8, 5, 8, 5, 4, 3, 9, 21, 78, 95, 62, 64, 57, 58, 594, 562, 55, 58, 65, 574, 582, 52, 56]

Merge Sort sorted output: [0, 3, 4, 5, 5, 8, 8, 9, 9, 21, 52, 55, 56, 57, 58, 58, 62, 64, 65, 78, 95, 562, 574, 582, 594]

Actual Count : 81

7.

Enter number of values 30

Enter Value 8

Enter Value 90

Enter Value 4

Enter Value 3

Enter Value 2

Enter Value 6

Enter Value 6

Enter Value 7

Enter Value 9

Enter Value 3

Enter Value 2

Enter Value 6

Enter Value 54

Enter Value 3

Enter Value 23

Enter Value 24

Enter Value 24

Enter Value 251

Enter Value 9

Enter Value 3

Enter Value 7

Enter Value 6

Enter Value 8

Enter Value 4

Enter Value 9

Enter Value 67

Enter Value 54

Enter Value 43

Enter Value 21

Enter Value 87

Unsorted data input [8, 90, 4, 3, 2, 6, 6, 7, 9, 3, 2, 6, 54, 3, 23, 24, 24, 251, 9, 3, 7, 6, 8, 4, 9, 67, 54, 43, 21, 87]

Merge Sort sorted output: [2, 2, 3, 3, 3, 3, 4, 4, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9, 9, 21, 23, 24, 24, 43, 54, 54, 67, 87, 90, 251]

Actual Count : 112

8.

Enter number of values 35

Enter Value 53

Enter Value 76

Enter Value 98

Enter Value 90

Enter Value 98

Enter Value 0

Enter Value 9

Enter Value 8

Enter Value 7

Enter Value 6

Enter Value 4

Enter Value 90

Enter Value 98

Enter Value 30

Enter Value 90

Enter Value 56

Enter Value 76

Enter Value 74

Enter Value 73

Enter Value 90

Enter Value 65

Enter Value 87

Enter Value 87

Enter Value 09

Enter Value 654

Enter Value 768

Enter Value 908

Enter Value 7654

Enter Value 432

Enter Value 98

Enter Value 76

Enter Value 54

Enter Value 54

Enter Value 32

Enter Value 99

Unsorted data input [53, 76, 98, 90, 98, 0, 9, 8, 7, 6, 4, 90, 98, 30, 90, 56, 76, 74, 73, 90, 65, 87, 87, 9, 654, 768, 908, 7654, 432, 98, 76, 54, 54, 32, 99]

Merge Sort sorted output: [0, 4, 6, 7, 8, 9, 9, 30, 32, 53, 54, 54, 56, 65, 73, 74, 76, 76, 76, 87, 87, 90, 90, 90, 90, 98, 98, 98, 98, 99, 432, 654, 768, 908, 7654]

Actual Count : 130

9.

Enter number of values 40

Enter Value 5

Enter Value 10

Enter Value 12

Enter Value 15

Enter Value 20

Enter Value 25

Enter Value 30

Enter Value 35

Enter Value 40

Enter Value 50

Enter Value 8

Enter Value 16

Enter Value 20

Enter Value 38

Enter Value 57

Enter Value 120

Enter Value 149

Enter Value 209

Enter Value 303

Enter Value 278

Enter Value 295

Enter Value 234

Enter Value 178

Enter Value 190

Enter Value 23

Enter Value 27

Enter Value 78

Enter Value 76

Enter Value 45

Enter Value 34

Enter Value 35

Enter Value 87

Enter Value 99

Enter Value 98

Enter Value 97

Enter Value 96

Enter Value 85

Enter Value 75

Enter Value 44

Enter Value 33

Unsorted data input [5, 10, 12, 15, 20, 25, 30, 35, 40, 50, 8, 16, 20, 38, 57, 120, 149, 209, 303, 278, 295, 234, 178, 190, 23, 27, 78, 76, 45, 34, 35, 87, 99, 98, 97, 96, 85, 75, 44, 33]

Merge Sort sorted output: [5, 8, 10, 12, 15, 16, 20, 20, 23, 25, 27, 30, 33, 34, 35, 35, 38, 40, 44, 45, 50, 57, 75, 76, 78, 85, 87, 96, 97, 98, 99, 120, 149, 178, 190, 209, 234, 278, 295, 303]

Actual Count : 146

10.

Enter number of values 50

Enter Value 50

Enter Value 49

Enter Value 48

Enter Value 47

Enter Value 46

Enter Value 45

Enter Value 43

Enter Value 42

Enter Value 41

Enter Value 40

Enter Value 39

Enter Value 38

Enter Value 36

Enter Value 35

Enter Value 34

Enter Value 32

Enter Value 31

Enter Value 30

Enter Value 29

Enter Value 28

Enter Value 27

Enter Value 26

Enter Value 25

Enter Value 24

Enter Value 23

Enter Value 21

Enter Value 20

Enter Value 19

Enter Value 18

Enter Value 17

Enter Value 16

Enter Value 15

Enter Value 14

Enter Value 13

Enter Value 12

Enter Value 11

Enter Value 10

Enter Value 9

Enter Value 8

Enter Value 7

Enter Value 6

Enter Value 5

Enter Value 4

Enter Value 3

Enter Value 2

Enter Value 1

Enter Value 0

Enter Value 32

Enter Value 33

Enter Value 34

Unsorted data input [50, 49, 48, 47, 46, 45, 43, 42, 41, 40, 39, 38, 36, 35, 34, 32, 31, 30, 29, 28, 27, 26, 25, 24, 23, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 32, 33, 34]

Merge Sort sorted output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 32, 33, 34, 34, 35, 36, 38, 39, 40, 41, 42, 43, 45, 46, 47, 48, 49, 50]

Actual Count: 157

**1.2.3 Output and Graphical Analysis Of Algorithm**

**1.2.3.1 INSERTION SORT**

**1.2.3.2 MERGE SORT**

**COMBINING RESULTS OF INSERTION SORT ALGORITHM AND MERGE SORT ALGORITHM**

**1.3 Source Programs**

**1.3.1 INSERTION SORT**

Implemented using PYTHON :

n = int(input("\n Enter number of values "))

count = 0

a = []

for i in range(n):

t = int(input("\n Enter Value "))

a.append(t)

print("\n Unsorted data input ",a)

for j in range(0,len(a)):

key = a[j]

i = j-1

while i>=0 and a[i]>key:

a[i+1]=a[i]

i = i-1

a[i+1]=key

count=count+1

print("\n Sorted data output ",a)

print("\n Actual Count ",count)

**1.3.2 MERGE SORT**

Implemented using PYTHON :

n = int(input("\n Enter number of values "))

a = []

count = 0

for i in range(n):

t = int(input("\n Enter Value "))

a.append(t)

print("\n Unsorted data input ",a)

def mergesort( a ):

\_mergesort( a, 0, len( a ) - 1 )

def \_mergesort( a, first, last ):

global count

# break problem into smaller structurally identical pieces

mid = ( first + last ) // 2

if first < last:

\_mergesort( a, first, mid )

\_mergesort( a, mid + 1, last )

# merge solved pieces to get solution to original problem

p, f, l = 0, first, mid + 1

tmp = [None] \* ( last - first + 1 )

while f <= mid and l <= last:

if a[f] < a[l] :

tmp[p] = a[f]

f =f+1

else:

tmp[p] = a[l]

l=l+1

p =p+1

count=count+1

if f <= mid :

tmp[p:] = a[f:mid + 1]

if l <= last:

tmp[p:] = a[l:last + 1]

p = 0

while first <= last:

a[first] = tmp[p]

first =first+1

p =p+1

mergesort(a)

print("\n Merge Sort sorted output:",a)

print("\n Actual Count :",count)

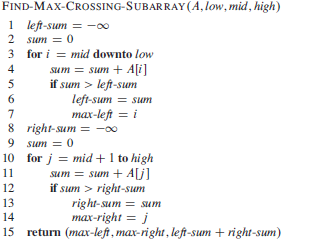
**CHAPTER 2**

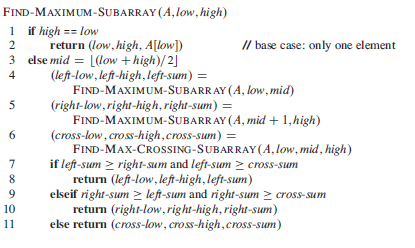
**MAXIMUM SUB-ARRAY PROBLEM**

**2.1 PROBLEM DESCRIPTION AND EXPLANATION**

The **maximum sub array problem** is the task of finding the contiguous sub array within a one-dimensional [array](http://en.wikipedia.org/wiki/Array_data_structure) of numbers (containing at least one positive number) which has the largest sum. For example, for the sequence of values −2, 1, −3, 4, −1, 2, 1, −5, 4; the contiguous sub array with the largest sum is 4, −1, 2, 1, with sum 6.

Divide and conquer method is adopted to solve the max sub array problem. According to this approach, the array is divided into two halves and the maximum sub array is found in each of the halves. Also the maximum sub array sum is calculated such that it crosses the midpoint. The time complexity for this approach is O (nLgn).





**2.2 TABULATION, OUTPUT AND ANALYSIS**

The Worst Case of maximum contiguous sub-array Algorithm is O(NLogN)

|  |  |  |
| --- | --- | --- |
| **N** | **Actual Count** | **O(nlogn)** |
| 16 | 18 | 19.2 |
| 20 | 22 | 26.0 |
| 25 | 32 | 34.9 |
| 35 | 47 | 54.04 |
| 45 | 53 | 74.35 |
| 55 | 72 | 95.71 |
|  |  |  |

**2.2.1 INPUT ARRAYS, SORTED ARRAY AND ANALYSIS**

Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:24:06) [MSC v.1600 32 bit (Intel)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================ RESTART ===================================>>>

1.

These are the values form Stocks analysis which is Text referenced

Original Array :=> [13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]

maxSum of the sub\_array :=> 43

starting interval of the max sub array => 7 ;

Ending interval of the max sub array => 10

Actual Count time complexity: 18

Remaining inputs:

2.

Original Array :=> [70, 93, 7, -97, -22, -40, -28, 67, -10, -25, -90, 29, 76, 46, -68, -73, -80, 21, -54, 0, -52, 13, 22, -14, 77, -43]

maxSum of the sub\_array :=> 170

starting interval of the max sub array => 0 ;

Ending interval of the max sub array => 2

Actual Count time complexity: 22.

3.

Original Array :=> [54, 36, -24, -84, 50, 29, 52, 55, -51, 22, 11, -87, 93, -92, -47, 9, -91, -67, -65, -99, 83, 92, 46, 79, 91, 74]

maxSum of the sub\_array :=> 465

Starting interval of the max sub array => 20 ;

Ending interval of the max sub array => 25

Actual Count time complexity: 32

4.

Original Array :=> [-72, 59, 56, 88, -91, -94, 25, -7, -23, 62, -18, 46, 45, 81, 87, -79, -50, 79, -40, -28, 84, -77, 72, 15, 26, 27, 38, -22, 68, 44, 41, -34, -19, -6, 75, -55]

maxSum of the sub\_array :=> 530

Starting interval of the max sub array => 1 ;

Ending interval of the max sub array => 34

Actual Count time complexity: 47

5.

Original Array :=> [-1, 20, -89, 89, -55, -10, 98, 82, 58, 93, -56, 73, 36, -94, -26, -17, 8, -6, -87, -22, 25, -96, 15, -40, 2, -97, 56, -61, 92, -64, -25, 42, 78, 51, -38, 33, 28, 90, -41, 52, -27, -28, 88, -31, -100, -81]

maxSum of the sub\_array :=> 408

Starting interval of the max sub array => 3 ;

Ending interval of the max sub array => 12

Actual Count time complexity: 53

6.

Original Array :=> [-50, -4, 23, 13, -61, -33, 3, 75, 11, -93, -1, -100, -27, 60, -67, 22, -98, 14, -23, -58, 16, 33, 69, 56, 63, 59, 94, -6, 10, 87, -43, -38, 96, 66, -8, 68, -16, -35, -44, 57, 47, -29, 44, -65, 0, -48, -78, -63, -80, 48, 92, -41, -22, -81, 30, -37]

maxSum of the sub\_array :=> 646

Starting interval of the max sub array => 20 ;

Ending interval of the max sub array => 42

Actual Count time complexity: 72

**2.2.3 OUTPUT WITH GRAPHICAL ANALYSIS OF ALGORITHM**

**2.3 SOURCE PROGRAM**

Implemented using PYTHON :

def maxSubArray(ls):

count1 = 0

count2 = 0

if len(ls) == 0:

raise Exception("Array empty") # should be non-empty

runSum = maxSum = ls[0]

i = 0

start = finish = 0

for j in range(1, len(ls)):

if ls[j] > (runSum + ls[j]):

runSum = ls[j]

i = j

else:

runSum += ls[j]

count1 = count1 + 1

if runSum > maxSum:

maxSum = runSum

start = i

finish = j

count2 = count2 + 1

print ("Original Array", ls)

print ("maxSum =>", maxSum)

print ("start =>", start, "; finish =>", finish)

print ("Actual Count :", count1+count2)

print ("\n")

print ("These are the values form Stocks analysis which is Text referenced")

maxSubArray([0, 13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7])

print ("Remaining inputs:")

print ("\n")

import random

ls = []

ls\_count =0

while ls\_count < 20:

number=random.randint(-100,100)

if number not in ls:

ls.append(number)

ls\_count += 1

#print(ls)

maxSubArray(ls)

import random

ls = []

ls\_count =0

while ls\_count < 25:

number=random.randint(-100,100)

if number not in ls:

ls.append(number)

ls\_count += 1

#print(ls)

maxSubArray(ls)

import random

ls = []

ls\_count =0

while ls\_count < 35:

number=random.randint(-100,100)

if number not in ls:

ls.append(number)

ls\_count += 1

#print(ls)

maxSubArray(ls)

import random

ls = []

ls\_count =0

while ls\_count < 45:

number=random.randint(-100,100)

if number not in ls:

ls.append(number)

ls\_count += 1

#print(ls)

maxSubArray(ls)

**CHAPTER 3**

**QUICK SORT OR RANDOMIZED QUICK SORT**

**3.1 PROBLEM DESCRIPTION AND EXPLANATION**

Quicksort is a fast sorting algorithm, which is used not only for educational purposes, but widely applied in practice. On the average, it has O(n log n) complexity, making quicksort suitable for sorting big data volumes. The idea of the algorithm is quite simple and once you realize it, you can write quicksort as fast as [bubble sort](http://www.algolist.net/Algorithms/Sorting/Bubble_sort).

## Algorithm

The divide-and-conquer strategy is used in quicksort. Below the recursion step is described:

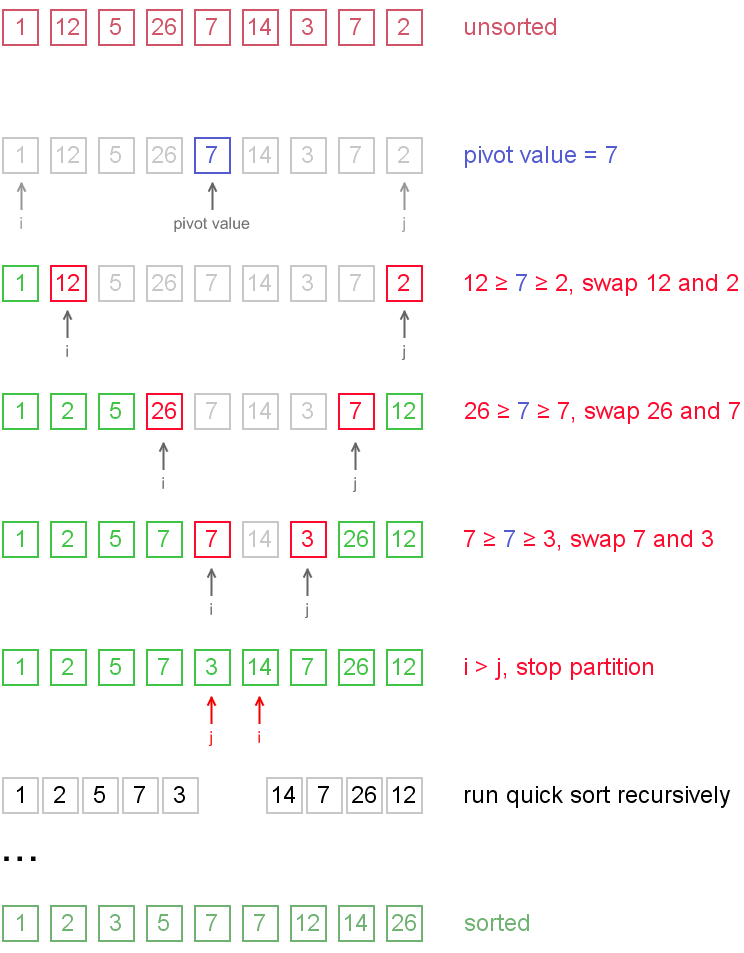
1. **Choose a pivot value.**We take the value of the middle element as pivot value, but it can be any value, which is in range of sorted values, even if it doesn't present in the array.
2. **Partition.**Rearrange elements in such a way, that all elements which are lesser than the pivot go to the left part of the array and all elements greater than the pivot, go to the right part of the array. Values equal to the pivot can stay in any part of the array. Notice, that array may be divided in non-equal parts.
3. **Sort both parts.**Apply quicksort algorithm recursively to the left and the right parts.

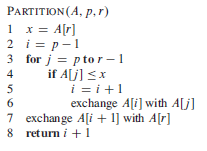
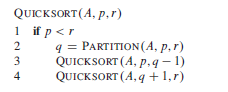
### Partition algorithm in detail

There are two indices **i** and **j** and at the very beginning of the partition algorithm **i** points to the first element in the array and **j** points to the last one. Then algorithm moves **i** forward, until an element with value greater or equal to the pivot is found. Index **j** is moved backward, until an element with value lesser or equal to the pivot is found. If **i ≤ j**then they are swapped and i steps to the next position (**i + 1**), j steps to the previous one **(j - 1)**. Algorithm stops, when**i** becomes greater than **j**.

After partition, all values before **i-th** element are less or equal than the pivot and all values after **j-th** element are greater or equal to the pivot.

Example.Sort {1, 12, 5, 26, 7, 14, 3, 7, 2} using quicksort.





**3.2 TABULATION OUTPUT AND ANALYSIS**

**Quick Sort**

Table of N: Actual count and Worst case O(n2)

|  |  |  |
| --- | --- | --- |
| **N** | **ACTUAL COUNT** | **WORST CASE O(n2)** |
| 7 | 6 | 49 |
| 12 | 13 | 144 |
| 15 | 16 | 225 |
| 17 | 18 | 289 |
| 20 | 23 | 400 |
| 25 | 30 | 625 |
| 30 | 35 | 900 |

**Randomized Quick Sort**

Table of N: Actual count and Worst case O(n lg n)

|  |  |  |
| --- | --- | --- |
| **N** | **Actual Count** | **O(nlogn)** |
| 15 | 10 | 17.64 |
| 20 | 23 | 26.02 |
| 25 | 30 | 34.94 |
| 35 | 48 | 54.04 |
| 45 | 64 | 74.39 |

Quick Sort Vs Randomized Quick Sort

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **Actual Count** | **O(n2)** | **Actual Count** | **O(n log n)** |
| 15 | 28 | 225 | 19 | 17.64 |
| 20 | 48 | 400 | 31 | 26.02 |
| 25 | 60 | 625 | 40 | 34.94 |
| 35 | 127 | 1225 | 62 | 54.04 |
| 45 | 182 | 1600 | 87 | 74.39 |

**3.2.1 INPUT ARRAYS, SORTED ARRAY AND ANALYSIS**

**3.2.1.1 Quick Sort**

1.

Enter 1 for manual input and 2 for random generated input:

1

Enter number of values:15

Enter the input values :

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

Beginning sorted data input:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

The Actual Time Complexity is 0

Ending sorted data output:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

The Actual Time Complexity is 28

2.

Enter 1 for manual input and 2 for random generated input:

1

Enter number of values:15

Enter the input values :

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

Beginning sorted data input:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

The Actual Time Complexity is 0

Ending sorted data output:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

The Actual Time Complexity is 48

3.

Enter 1 for manual input and 2 for random generated input:

1

Enter number of values:25

Enter the input values :

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

Beginning sorted data input:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

The Actual Time Complexity is 0

Ending sorted data output:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

The Actual Time Complexity is 60

4.

Enter 1 for manual input and 2 for random generated input:

1

Enter number of values:35

Enter the input values :

35

34

33

32

31

30

29

28

27

26

25

24

23

22

21

20

19

18

17

16

15

14

13

12

11

10

9

8

7

6

5

4

3

2

1

Beginning sorted data input:

35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9

8 7 6 5 4 3 2 1

The Actual Time Complexity is 0

Ending sorted data output:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

31 32 33 34 35

The Actual Time Complexity is 127

5.

Enter 1 for manual input and 2 for random generated input:

1

Enter number of values:35

Enter the input values :

45

44

43

42

41

40

39

38

37

36

35

34

33

32

31

30

29

28

27

26

25

24

23

22

21

20

19

18

17

16

15

14

13

12

11

10

9

8

7

6

5

4

3

2

1

Beginning sorted data input:

45 44 43 42 41 4039 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9

8 7 6 5 4 3 2 1

The Actual Time Complexity is 0

Ending sorted data output:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

31 32 33 34 35 36 37 38 39 40 41 42 43 44 45

The Actual Time Complexity is 182

**3.2.1.2 Randomized Quick Sort**

1.

Enter 1 for manual input and 2 for random generated input:

2

Enter number of values: 15

Beginning Randomized data input:

26 133 47 133 44 49 80 54 190 118 42 175 82 142 178

Ending sorted Random data output:

26 42 44 47 49 54 80 82 118 133 133 142 175 178 190

The Actual Time Complexity is 10

2.

Enter 1 for manual input and 2 for random generated input:

2

Enter number of values: 20

Beginning Randomized data input:

39 3 148 24 99 120 67 103 7 161 127 29 53 127 68 30 150 141 45 158

Ending sorted Random data output:

3 7 24 29 30 39 45 53 67 68 99 103 120 127 127 141 148 150 158 161

The Actual Time Complexity is 23

3.

Enter 1 for manual input and 2 for random generated input:

2

Enter number of values: 25

Beginning Randomized data input:

179 32 43 171 41 129 31 102 77 138 167 123 153 123 59 72 44 25 62 111 36 79 112

165 78

Ending sorted Random data output:

25 31 32 36 41 43 44 59 62 72 77 78 79 102 111 112 123 123 129 138 153 165 167 1

71 179

The Actual Time Complexity is 30

4.

Enter 1 for manual input and 2 for random generated input:

2

Enter number of values: 35

Beginning Randomized data input:

152 42 172 174 58 70 151 197 55 170 128 58 65 110 196 87 5 90 186 134 34 168 66

122 4 102 45 110 55 58 188 91 111 198 22

Ending sorted Random data output:

4 5 22 34 42 45 55 55 58 58 58 65 66 70 87 90 91 102 110 110 111 122 128 134 151

152 168 170 172 174 186 188 196 197 198

The Actual Time Complexity is 48

5.

Enter 1 for manual input and 2 for random generated input:

2

Enter number of values: 45

Beginning Randomized data input:

102 148 124 175 8 59 51 14 124 103 125 84 46 9 27 39 17 82 64 125 59 164 20 82 1

80 118 166 127 179 67 160 109 146 29 6 90 108 69 143 147 43 55 16 83 70

Ending sorted Random data output:

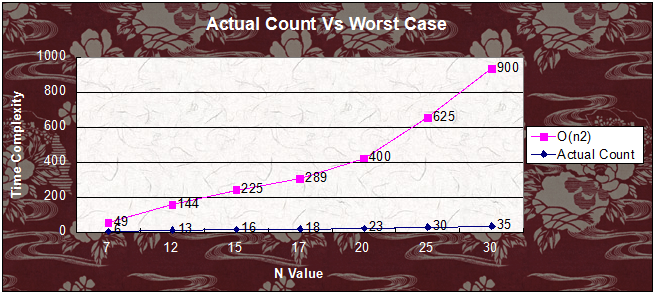
6 8 9 14 16 17 20 27 29 39 43 46 51 55 59 59 64 67 69 70 82 82 83 84 90 102 103

108 109 118 124 124 125 125 127 143 146 147 148 160 164 166 175 179 180

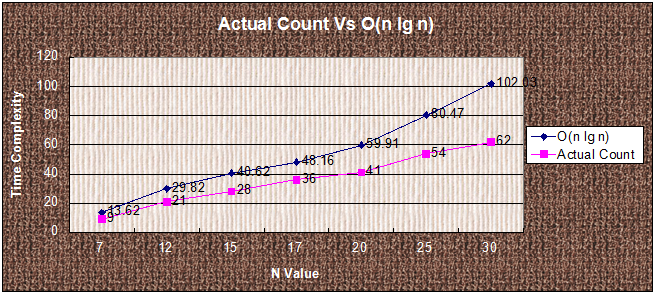
The Actual Time Complexity is 64

**3.2.3 OUTPUT AND GRAPHICAL ANALYSIS OF ALGORITHM**

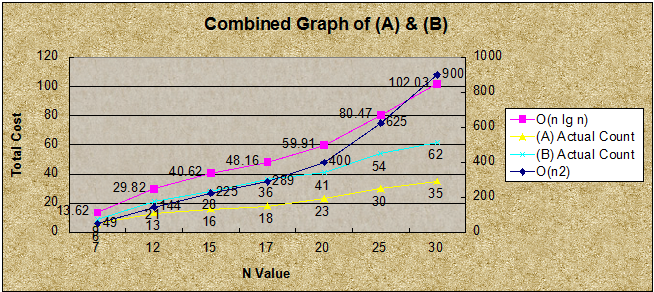
**3.2.3.1 Quick Sort**



**3.2.3.2 Randomized Quick Sort**



**COMBINED GRAPH OF (A) AND (B)**



**3.3 SOURCE PROGRAM**

Implemented using c++:

#include <iostream>

# include <ctime>

#include <cstdlib>

using namespace std;

int counter =0;

void display(int arr1[], int size)

{

for (int i =1; i <=size ; i++)

{

cout << arr1[i] << " ";

}

cout<<"\n\n The Actual Time Complexity is "<<counter;

}

void display2(int arr1[], int size)

{

for (int i =0; i <=size ; i++)

{

cout << arr1[i] << " ";

}

cout<<"\n\n The Actual Time Complexity is "<<counter;

}

void quickSort(int arr[], int left, int right)

{

int i = left, j = right;

int tmp;

int pivot = arr[(left + right) / 2];

/\* partition module\*/

while (i <= j)

{

while (arr[i] < pivot)

i++;

counter++;

while (arr[j] > pivot)

j--;

counter++;

if (i <= j)

{

tmp = arr[i];

arr[i] = arr[j];

arr[j] = tmp;

i++;

j--;

}

counter++;

};

/\* recursion module \*/

if (left < j)

quickSort(arr, left, j);

if (i < right)

quickSort(arr, i, right);

counter ++;

}

int main()

{

int ch;

cout << "Enter 1 for manual input and 2 for random generated input:\n";

cin >> ch;

switch (ch)

{

case 1 :

int num ;

cout << "Enter number of values:";

cin >> num;

int arr[num];

cout << "Enter the input values :" << endl;

for (int i = 1; i <= num ; i++)

{

cin >> arr[i];

}

cout << "Beginning sorted data input :" << endl;

display(arr , num);

cout << endl << endl;

quickSort(arr,1,num);

cout << "Ending sorted data output :" << endl;

display(arr,num);

break;

case 2:

int n;

cout << "Enter number of values:";

cin >> n;

int a[n];

srand((unsigned)time(0));

cout << "Beginning Randomized data input :" << endl;

for (int i =0 ; i < n ;i++)

{

a[i] = (rand()%200)+1;

cout << a[i] << " ";

}

cout << endl << endl;

quickSort(a,0,n-1);

cout << "Ending sorted data random output :" << endl;

display2(a,n-1);

}

return 0;

}

**CHAPTER 4**

**HEAP SORT**

**4.1 PROBLEM DESCRIPTION AND EXPLANATION**

The **(*binary*) *heap*** data structure is an array object that can be viewed as a complete binary tree. Each node of the tree corresponds to an element of the array that stores the value in the node. The tree is completely filled on all levels except possibly the lowest, which is filled from the left up to a point. An array *A* that represents a heap is an object with two attributes:*length*[*A*], which is the number of elements in the array, and *heap-size*[*A*], the number of elements in the heap stored within array *A*. That is, although *A*[1 . . *length*[*A*]] may contain valid numbers, no element past *A*[*heap-size*[*A*]], where *heap-size*[*A*] [*length*[*A*], is an element of the heap. The root of the tree is *A*[1], and given the index *i* of a node, the indices of its parent PARENT(*i*), left child LEFT(*i*), and right child RIGHT(*i*) can be computed simply:



PARENT(*i*)  
**return** *i*/2



LEFT(*i*)  
**return** 2*i*

RIGHT(*i*)  
**return** 2*i* + 1

HEAPIFY is an important subroutine for manipulating heaps. Its inputs are an array *A* and an index *i* into the array. When HEAPIFY is called, it is assumed that the binary trees rooted at LEFT(*i*) and RIGHT(*i*) are heaps, but that *A*[*i*] may be smaller than its children, thus violating the heap property (7.1). The function of HEAPIFY is to let the value at *A*[*i*] "float down" in the heap so that the subtree rooted at index *i*becomes a heap.

HEAPIFY(*A*, *i*)  
1 *l* LEFT(*i*)  
2 *r* RIGHT(*i*)  
3 **if** *l* *heap-size*[*A*] and *A*[*l*] > *A*[*i*]  
4 **then** largest l  
5 **else** *largest* *i*6 **if** *r* *heap-size*[*A*] and *A*[*r*] > *A*[*largest*]  
7 **then** *largest* *r*8 **if** *largest* *i*9 **then** exchange *A*[*i*] *A*[*largest*]  
10 HEAPIFY(*A,largest*)



The procedure BUILD-HEAP goes through the remaining nodes of the tree and runs HEAPIFY on each one. The order in which the nodes are processed guarantees that the sub trees rooted at children of a node *i* are heaps before HEAPIFY is run at that node.

BUILD-HEAP(*A*)  
1 *heap-size*[*A*] *length*[*A*]  
2 **for** *i* *length*[*A*]/2**downto** 1  
3 **do** HEAPIFY(*A, i*)



The heapsort algorithm starts by using BUILD-HEAP to build a heap on the input array *A*[1 . . *n*], where *n* = *length*[*A*]. Since the maximum element of the array is stored at the root *A*[1], it can be put into its correct final position by exchanging it with *A*[*n*]. If we now "discard" node *n* from the heap (by decrementing *heap-size*[*A*]), we observe that *A*[1 *. .* (*n -* 1)]can easily be made into a heap. The children of the root remain heaps, but the new root element may violate the heap property (7.1). All that is needed to restore the heap property, however, is one call to HEAPIFY(*A*, 1), which leaves a heap in *A*[1 *. .* (*n -* 1)]. The heapsort algorithm then repeats this process for the heap of size *n* - 1 down to a heap of size 2.

HEAPSORT(*A*)  
1 BUILD-HEAP(*A*)  
2 **for***i* *length*[*A*] **downto** 2  
3 **do** exchange *A*[1] *A*[i]  
4 *heap-size*[*A*] *heap-size*[*A*] -1  
5 HEAPIFY(*A*, 1)

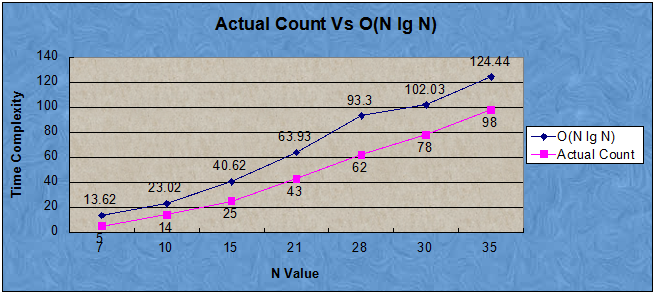


**4.2 TABULATION OUTPUT AND ANALYSIS**

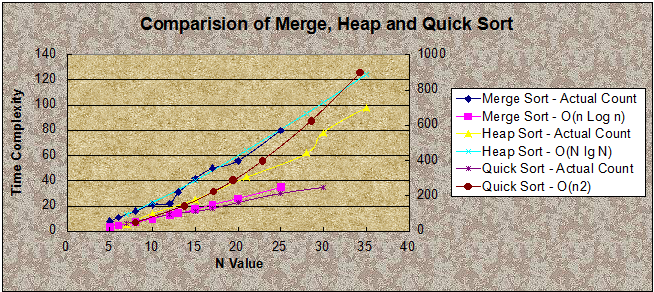
**4.2.1 INPUT ARRAYS, SORTED ARRAY AND ANALYSIS**

**4.2.2 WITH MORE VALUES**

**4.2.3 OUTPUT AND GRAPHICAL ANALYSIS OF ALGORITHM**



**4.2.4 GRAPHICAL COMPARISION OF MERGE, HEAP AND QUICK SORTS**



**4.3 SOURCE PROGRAM**

**CHAPTER 5**

**HASHING AND HASH TABLES**

**5.1 PROBLEM DESCRIPTION AND EXPLANATION**

A **hash table**, or a **hash map**, is a data structure that associates *keys* with *values*. The primary operation it supports efficiently is a *lookup*: given a key (e.g. a person's name), find the corresponding value (e.g. that person's telephone number). It works by transforming the key using a [hash function](https://en.wikipedia.org/wiki/hash_function) into a *hash*, a number that the hash table uses to locate the desired value. This hash maps directly to a bucket in the array of key/value pairs, hence the name hash map. The mapping method lets us directly access the storage location for any key/value pair.

**Hash table<Element> Operations**

make-hash-table(integer *n*): HashTable

Create a hash table with *n* buckets.

get-value(HashTable *h*, Comparable *key*): Element

Returns the value of the element for the given *key*. The *key* must be some comparable type.

set-value(HashTable *h*, Comparable *key*, Element *new-value*)

Sets the element of the array for the given *key* to be equal to *new-value*. The *key* must be some comparable type.

remove(HashTable *h*, Comparable *key*)

Remove the element for the given *key* from the hash table. The *key* must be some comparable type.

**Hash Functions** There are three hash functions that can be used to create hash tables.  
**(a)** **The division method**In the ***division method*** for creating hash functions, we map a key k into one of m slots by taking the remainder of k divided by m.   
That is, the hash function is h(k) = k mod m

**(b) The multiplication method**The ***multiplication method*** for creating hash functions operates in two steps. First, we multiply the key k by a constant A in the range 0 < A < 1 and extract the fractional part of kA. Then, we multiply this value by m and take the floor of the result.   
In short, the hash function is h(k) = L m(k A mod)˩, where A ≈ (√5 - 1) / 2 = 0:6180339887

**(c) Quadratic probing**

***Quadratic probing*** uses a hash function of the form

h(k, i) = (h’(k) + c1i + c2i2 ) mod m ,

where h0 is an auxiliary hash function, c1 and c2 are positive auxiliary constants, and i D 0; 1 . .. . ..m - 1. The initial position probed is T [h’(k)] later positions probed are offset by amounts that depend in a quadratic manner on the probe number i . This method works much better than linear probing, but to make full use of the hash table, the values of c1, c2, and m are constrained. Also, if two keys have the same initial probe position, then their probe sequences are the same, since h(k1, 0) = h(k2, 0) implies h(k1, i) = h(k2, i). This property leads to a milder form of clustering, called ***secondary clustering***. As in linear probing, the initial probe determines the entire sequence, and so only m distinct probe sequences are used.

**5.2 TABULATION, OUTPUT AND ANALYSIS**

**5.2.1 Division Method**

Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART================================

>>>

**Enter total number of ID’s** 150

**Enter hashing value** 97

**150 4-digit student ID's are:**

[8962, 1660, 1423, 4189, 2621, 6895, 1460, 5391, 3373, 6207, 7040, 5704, 5224, 9285, 3792, 4340, 9615, 5365, 9275, 4343, 6104, 2257, 2032, 7748, 8272, 1689, 6830, 4916, 9384, 2669, 6610, 6693, 5366, 1123, 2837, 5372, 8557, 8166, 2733, 5756, 7508, 5717, 4788, 6955, 6770, 6696, 4184, 8249, 4203, 1717, 8605, 4561, 2163, 1601, 5018, 5758, 9206, 9809, 6675, 8545, 3699, 5840, 8189, 3917, 6987, 8200, 4456, 3621, 5494, 3073, 2150, 3420, 7111, 1219, 5359, 1530, 1360, 8058, 5708, 6158, 1712, 5279, 8699, 8947, 4400, 9436, 2070, 2544, 8222, 8491, 8234, 8179, 2846, 9983, 6954, 1781, 7403, 4006, 9315, 8353, 4512, 8082, 1691, 9870, 8357, 6579, 5752, 1372, 1445, 2459, 7947, 4260, 2017, 3954, 7911, 2708, 3673, 1099, 9278, 5696, 2020, 1187, 9711, 5998, 2745, 9560, 6566, 7214, 1591, 7478, 8940, 9253, 9464, 4398, 1560, 1847, 8287, 2804, 9213, 1065, 8432, 1976, 9046, 8916, 7584, 9876, 1018, 5112, 5647, 9656]

**Mapped hash index to student ID's:**

[38, 11, 65, 18, 2, 8, 5, 56, 75, 96, 56, 78, 83, 70, 9, 72, 12, 30, 60, 75, 90, 26, 92, 85, 27, 40, 40, 66, 72, 50, 14, 0, 31, 56, 24, 37, 21, 18, 17, 33, 39, 91, 35, 68, 77, 3, 13, 4, 32, 68, 69, 2, 29, 49, 71, 35, 88, 12, 79, 9, 13, 20, 41, 37, 3, 52, 91, 32, 62, 66, 16, 25, 30, 55, 24, 75, 2, 7, 82, 47, 63, 41, 66, 23, 35, 27, 33, 22, 74, 52, 86, 31, 33, 89, 67, 35, 31, 29, 3, 11, 50, 31, 42, 73, 15, 80, 29, 14, 87, 34, 90, 89, 77, 74, 54, 89, 84, 32, 63, 70, 80, 23, 11, 81, 29, 54, 67, 36, 39, 9, 16, 38, 55, 33, 8, 4, 42, 88, 95, 95, 90, 36, 25, 89, 18, 79, 48, 68, 21, 53]

**Hash table with chain values for each cell:**

0 --> [6693]

2 --> [2621, 4561, 1360]

3 --> [6696, 6987, 9315]

4 --> [8249, 1847]

5 --> [1460]

7 --> [8058]

8 --> [6895, 1560]

9 --> [3792, 8545, 7478]

11 --> [1660, 8353, 9711]

12 --> [9615, 9809]

13 --> [4184, 3699]

14 --> [6610, 1372]

15 --> [8357]

16 --> [2150, 8940]

17 --> [2733]

18 --> [4189, 8166, 7584]

20 --> [5840]

21 --> [8557, 5647]

22 --> [2544]

23 --> [8947, 1187]

24 --> [2837, 5359]

25 --> [3420, 9046]

26 --> [2257]

27 --> [8272, 9436]

29 --> [2163, 4006, 5752, 2745]

30 --> [5365, 7111]

31 --> [5366, 8179, 7403, 8082]

32 --> [4203, 3621, 1099]

33 --> [5756, 2070, 2846, 4398]

34 --> [2459]

35 --> [4788, 5758, 4400, 1781]

36 --> [7214, 1976]

37 --> [5372, 3917]

38 --> [8962, 9253]

39 --> [7508, 1591]

40 --> [1689, 6830]

41 --> [8189, 5279]

42 --> [1691, 8287]

47 --> [6158]

48 --> [1018]

49 --> [1601]

50 --> [2669, 4512]

52 --> [8200, 8491]

53 --> [9656]

54 --> [7911, 9560]

55 --> [1219, 9464]

56 --> [5391, 7040, 1123]

60 --> [9275]

62 --> [5494]

63 --> [1712, 9278]

65 --> [1423]

66 --> [4916, 3073, 8699]

67 --> [6954, 6566]

68 --> [6955, 1717, 5112]

69 --> [8605]

70 --> [9285, 5696]

71 --> [5018]

72 --> [4340, 9384]

73 --> [9870]

74 --> [8222, 3954]

75 --> [3373, 4343, 1530]

77 --> [6770, 2017]

78 --> [5704]

79 --> [6675, 9876]

80 --> [6579, 2020]

81 --> [5998]

82 --> [5708]

83 --> [5224]

84 --> [3673]

85 --> [7748]

86 --> [8234]

87 --> [1445]

88 --> [9206, 2804]

89 --> [9983, 4260, 2708, 8916]

90 --> [6104, 7947, 8432]

91 --> [5717, 4456]

92 --> [2032]

95 --> [9213, 1065]

96 --> [6207]

**Enter 1st Student ID:** 8353

**Enter 2nd Student ID:** 1445

**Enter 3rd Student ID:** 10000

**1st Student ID found at Hash index 11**

**2nd Student ID found at Hash index 87**

**3rd Student ID not found**

**5.2.2 Multiplication Method**

Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

**Enter total number of ID’s** 150

**Enter hashing value** 97

**150 4-digit student ID's are:**

[8973, 5149, 3850, 8344, 7492, 4078, 3666, 8249, 1929, 2804, 1180, 9837, 8805, 9715, 9993, 5346, 8646, 5096, 1063, 6306, 2576, 4645, 2865, 6425, 7702, 9914, 5284, 4405, 9373, 5850, 9097, 3328, 1680, 7754, 1745, 2370, 4499, 1283, 4332, 7376, 4174, 4139, 1328, 4927, 9821, 8228, 5671, 1540, 6735, 7541, 5043, 8539, 1987, 8604, 3279, 2951, 4051, 1245, 9257, 7097, 3047, 1499, 7972, 3116, 2042, 9419, 7793, 1729, 9660, 4391, 7168, 5066, 7089, 1786, 2173, 9433, 8046, 6960, 3287, 1382, 5169, 3066, 1044, 7868, 4442, 2490, 7823, 7337, 8162, 9718, 3167, 9243, 6531, 4732, 4475, 9498, 3660, 1168, 1202, 2400, 5496, 1324, 4827, 6454, 4552, 1506, 9591, 5269, 3232, 6341, 2879, 7407, 7203, 4546, 4057, 3837, 3889, 4738, 3378, 1081, 3107, 5472, 8351, 4794, 8918, 2079, 9837, 7440, 2648, 2265, 7161, 2998, 1740, 8380, 2995, 2020, 8276, 4137, 9984, 3118, 3097, 1876, 6382, 8101, 2213, 3430, 9753, 8473, 5151, 8038]

**Mapped hash index to student ID's:**

[60, 24, 41, 84, 30, 33, 69, 15, 18, 93, 27, 58, 76, 19, 1, 0, 50, 48, 94, 31, 5, 74, 64, 84, 9, 18, 67, 42, 80, 48, 24, 79, 28, 22, 45, 71, 51, 90, 31, 60, 65, 4, 72, 5, 69, 17, 84, 74, 44, 57, 72, 38, 3, 54, 51, 79, 63, 43, 13, 18, 14, 41, 93, 77, 2, 25, 32, 56, 20, 76, 6, 93, 23, 78, 95, 88, 68, 50, 46, 11, 59, 86, 22, 67, 29, 87, 85, 49, 38, 5, 30, 47, 36, 52, 68, 8, 0, 83, 85, 27, 69, 26, 24, 76, 28, 73, 54, 40, 47, 92, 31, 75, 67, 56, 35, 38, 51, 23, 69, 9, 22, 85, 19, 82, 60, 86, 58, 16, 53, 82, 71, 83, 36, 12, 1, 41, 82, 78, 43, 2, 4, 41, 28, 67, 68, 83, 66, 58, 47, 73]

**Hash table with chain values for each cell:**

0 --> [5346, 3660]

1 --> [9993, 2995]

2 --> [2042, 3118]

3 --> [1987]

4 --> [4139, 3097]

5 --> [2576, 4927, 9718]

6 --> [7168]

8 --> [9498]

9 --> [7702, 1081]

11 --> [1382]

12 --> [8380]

13 --> [9257]

14 --> [3047]

15 --> [8249]

16 --> [7440]

17 --> [8228]

18 --> [1929, 9914, 7097]

19 --> [9715, 8351]

20 --> [9660]

22 --> [7754, 1044, 3107]

23 --> [7089, 4738]

24 --> [5149, 9097, 4827]

25 --> [9419]

26 --> [1324]

27 --> [1180, 2400]

28 --> [1680, 4552, 6382]

29 --> [4442]

30 --> [7492, 3167]

31 --> [6306, 4332, 2879]

32 --> [7793]

33 --> [4078]

35 --> [4057]

36 --> [6531, 1740]

38 --> [8539, 8162, 3837]

40 --> [5269]

41 --> [3850, 1499, 2020, 1876]

42 --> [4405]

43 --> [1245, 9984]

44 --> [6735]

45 --> [1745]

46 --> [3287]

47 --> [9243, 3232, 5151]

48 --> [5096, 5850]

49 --> [7337]

50 --> [8646, 6960]

51 --> [4499, 3279, 3889]

52 --> [4732]

53 --> [2648]

54 --> [8604, 9591]

56 --> [1729, 4546]

57 --> [7541]

58 --> [9837, 9837, 8473]

59 --> [5169]

60 --> [8973, 7376, 8918]

63 --> [4051]

64 --> [2865]

65 --> [4174]

66 --> [9753]

67 --> [5284, 7868, 7203, 8101]

68 --> [8046, 4475, 2213]

69 --> [3666, 9821, 5496, 3378]

71 --> [2370, 7161]

72 --> [1328, 5043]

73 --> [1506, 8038]

74 --> [4645, 1540]

75 --> [7407]

76 --> [8805, 4391, 6454]

77 --> [3116]

78 --> [1786, 4137]

79 --> [3328, 2951]

80 --> [9373]

82 --> [4794, 2265, 8276]

83 --> [1168, 2998, 3430]

84 --> [8344, 6425, 5671]

85 --> [7823, 1202, 5472]

86 --> [3066, 2079]

87 --> [2490]

88 --> [9433]

90 --> [1283]

92 --> [6341]

93 --> [2804, 7972, 5066]

94 --> [1063]

95 --> [2173]

**Enter 1st Student ID:** 4442

**Enter 2nd Student ID:** 3430

**Enter 3rd Student ID:** 999

**1st Student ID found at Hash index 29**

**2nd Student ID found at Hash index 83**

**3rd Student ID not found**

**5.2.3 Quadratic Probing**

Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

**Enter total number of Id’s** 60

**Enter hashing value** 97

**60 4-digit student ID's are:**

[6504, 5855, 2257, 7769, 3006, 7412, 4109, 7842, 2550, 4348, 6900, 2523, 6885, 7143, 9024, 7327, 3941, 3163, 7940, 5569, 8392, 4406, 4821, 5839, 2959, 8266, 3837, 2556, 7860, 8879, 8457, 1852, 9254, 1343, 2359, 6334, 6897, 4194, 6964, 7954, 9495, 9398, 9387, 5423, 6716, 6344, 3021, 6468, 5568, 8872, 4664, 3940, 3136, 8347, 2262, 9520, 9987, 4119, 3576, 2398]

**Mapped hash index to student ID's:**

[5, 35, 26, 9, 96, 40, 35, 82, 28, 80, 13, 1, 95, 62, 3, 52, 61, 59, 83, 40, 50, 41, 68, 19, 49, 21, 54, 34, 3, 52, 18, 9, 39, 82, 31, 29, 10, 23, 77, 0, 86, 86, 75, 88, 23, 39, 14, 66, 39, 45, 8, 60, 32, 5, 31, 14, 93, 45, 84, 70]

**Hash table with chain values for each cell:**

0 --> [7954]

1 --> [2523]

3 --> [9024, 7860]

5 --> [6504, 8347]

8 --> [4664]

9 --> [7769, 1852]

10 --> [6897]

13 --> [6900]

14 --> [3021, 9520]

18 --> [8457]

19 --> [5839]

21 --> [8266]

23 --> [4194, 6716]

26 --> [2257]

28 --> [2550]

29 --> [6334]

31 --> [2359, 2262]

32 --> [3136]

34 --> [2556]

35 --> [5855, 4109]

39 --> [9254, 6344, 5568]

40 --> [7412, 5569]

41 --> [4406]

45 --> [8872, 4119]

49 --> [2959]

50 --> [8392]

52 --> [7327, 8879]

54 --> [3837]

59 --> [3163]

60 --> [3940]

61 --> [3941]

62 --> [7143]

66 --> [6468]

68 --> [4821]

70 --> [2398]

75 --> [9387]

77 --> [6964]

80 --> [4348]

82 --> [7842, 1343]

83 --> [7940]

84 --> [3576]

86 --> [9495, 9398]

88 --> [5423]

93 --> [9987]

95 --> [6885]

96 --> [3006]

**Enter 1st Student ID:** 4109

**Enter 2nd Student ID:** 3576

**Enter 3rd Student ID:** 2000

**1st Student ID found at Hash index 35**

**2nd Student ID found at Hash index 84**

**3rd Student ID not found**

**5.3 SOURCE PROGRAMS**

Implemented in Python programming language.

**5.3.1 DIVISION METHOD**

import os

import random

data = []

t = []

n = int(input("\n Enter total number of ID’s "))

m = int(input("\n Enter hashing value "))

for i in range(n):

Data.append(random.randint(1000, 9999))

print("\n 150 4-digit student ID's are: ")

print("\n", data)

l = len(data)

#print(l)

def hfunc(data, m):

return list(map(lambda index : index % m, data)) # Division Method

def htable(table):

table = [None]\*m

return table

def get\_index(keys, m):

index = hfunc(keys, m)

return(index)

def get\_i(index):

for i in index:

return(i)

def get\_k(data):

for k in data:

return k

#print ("\n", get\_index(data, m))

def linear(keys):

table = htable(m)

index = get\_index(keys, m)

j = 0

i = get\_i(index)

k = get\_k(keys)

while (j == m):

if table[i] == None:

table[i] = k

j = j+1

x = []

x = get\_index(data, m)

print("\n Mapped hash index to student ID's: ")

print ("\n", x)

output = []

seen = set()

for value in x:

# If value has not been encountered yet,

# ... add it to both list and set.

if value not in seen:

output.append(value)

seen.add(value)

output.sort()

for p in range(0,150):

q = (data[p],x[p])

t.append(q)

values = set(map(lambda x:x[1], t))

newlist = [[y[0] for y in t if y[1]==x] for x in values]

print("\n Hash table with chain values for each cell: ")

for w in range(0,len(output)):

print("\n",output[w],"-->",newlist[w])

s1 = int(input("\n Enter 1st Student ID: "))

s2 = int(input("\n Enter 2nd Student ID: "))

s3 = int(input("\n Enter 3rd Student ID: "))

if s1 in data:

i1 = data.index(s1)

print("\n 1st Student ID found at Hash index ",x[i1])

else:

print("\n 1st Student ID not found")

if s2 in data:

i2 = data.index(s2)

print("\n 2nd Student ID found at Hash index ",x[i2])

else:

print("\n 2nd Student ID not found")

if s3 in data:

i3 = data.index(s3)

print("\n 3rd Student ID found at Hash index ",x[i3])

else:

print("\n 3rd Student ID not found")

os.system("pause")

**5.3.2 MULTIPLICATION METHOD**

import os

import random

import math

data = []

t = []

n = int(input("\n Enter total number of ID’s"))

m = int(input("\n Enter hashing value "))

for i in range(n):

Data.append(random.randint(1000, 9999))

print("\n 150 4-digit student ID's are: ")

print("\n", data)

l = len(data)

#print(l)

def hfunc(data, m):

a = (math.sqrt(5)-1)/2

return list(map(lambda index : math.floor(m\*(index\*a % 1)), data)) # Multiplication Method

def htable(table):

table = [None]\*m

return table

def get\_index(keys, m):

index = hfunc(keys, m)

return(index)

def get\_i(index):

for i in index:

return(i)

def get\_k(data):

for k in data:

return k

#print ("\n", get\_index(data, m))

def linear(keys):

table = htable(m)

index = get\_index(keys, m)

j = 0

i = get\_i(index)

k = get\_k(keys)

while (j == m):

if table[i] == None:

table[i] = k

j = j+1

x = []

x = get\_index(data, m)

print("\n Mapped hash index to student ID's: ")

print ("\n", x)

output = []

seen = set()

for value in x:

# If value has not been encountered yet,

# ... add it to both list and set.

if value not in seen:

output.append(value)

seen.add(value)

output.sort()

for p in range(0,150):

q = (data[p],x[p])

t.append(q)

values = set(map(lambda x:x[1], t))

newlist = [[y[0] for y in t if y[1]==x] for x in values]

print("\n Hash table with chain values for each cell: ")

for w in range(0,len(output)):

print("\n",output[w],"-->",newlist[w])

s1 = int(input("\n Enter 1st Student ID: "))

s2 = int(input("\n Enter 2nd Student ID: "))

s3 = int(input("\n Enter 3rd Student ID: "))

if s1 in data:

i1 = data.index(s1)

print("\n 1st Student ID found at Hash index ",x[i1])

else:

print("\n 1st Student ID not found")

if s2 in data:

i2 = data.index(s2)

print("\n 2nd Student ID found at Hash index ",x[i2])

else:

print("\n 2nd Student ID not found")

if s3 in data:

i3 = data.index(s3)

print("\n 3rd Student ID found at Hash index ",x[i3])

else:

print("\n 3rd Student ID not found")

os.system("pause")

**5.3.3 QUADRATIC PROBING**

import os

import random

Import math

data = []

t = []

n = int(input("\n Enter total number of ID’s"))

m = int(input("\n Enter hashing value "))

for i in range(n):

Data.append(random.randint(1000, 9999))

print("\n 60 4-digit student ID's are: ")

print("\n", data)

l = len(data)

#print(l)

def hfunc(data, m):

for i in range(0,60):

return list(map(lambda index : math.floor((index % m + i\*i) % m), data)) # Quadratic Probing

def htable(table):

table = [None]\*m

return table

def get\_index(keys, m):

index = hfunc(keys, m)

return(index)

def get\_i(index):

for i in index:

return(i)

def get\_k(data):

for k in data:

return k

#print ("\n", get\_index(data, m))

def linear(keys):

table = htable(m)

index = get\_index(keys, m)

j = 0

i = get\_i(index)

k = get\_k(keys)

while (j == m):

if table[i] == None:

table[i] = k

j = j+1

x = []

x = get\_index(data, m)

print("\n Mapped hash index to student ID's: ")

print ("\n", x)

output = []

seen = set()

for value in x:

# If value has not been encountered yet,

# ... add it to both list and set.

if value not in seen:

output.append(value)

seen.add(value)

output.sort()

for p in range(0,60):

q = (data[p],x[p])

t.append(q)

values = set(map(lambda x:x[1], t))

newlist = [[y[0] for y in t if y[1]==x] for x in values]

print("\n Student ID’s: ")

for w in range(0,len(newlist)):

print("\n",newlist[w])

s1 = int(input("\n Enter 1st Student ID: "))

s2 = int(input("\n Enter 2nd Student ID: "))

s3 = int(input("\n Enter 3rd Student ID: "))

if s1 in data:

i1 = data.index(s1)

print("\n 1st Student ID found at Hash index ",x[i1])

else:

print("\n 1st Student ID not found")

if s2 in data:

i2 = data.index(s2)

print("\n 2nd Student ID found at Hash index ",x[i2])

else:

print("\n 2nd Student ID not found")

if s3 in data:

i3 = data.index(s3)

print("\n 3rd Student ID found at Hash index ",x[i3])

else:

print("\n 3rd Student ID not found")

os.system("pause")

**CHAPTER 6**

**HUFFMAN CODES PROBLEM AND GREEDY ALGORITHMS**

**6 PROBLEM DESCRIPTION AND EXPLANATION**

Huffman invented a greedy algorithm that constructs an optimal prefix code called a ***Huffman code***. In line with our observations in Section 16.2, its proof of correctness relies on the greedy-choice property and optimal substructure. Rather than demonstrating that these properties hold and then developing pseudocode, we present the pseudocode first. Doing so will help clarify how the algorithm makes greedy choices. In the pseudocode that follows, we assume that C is a set of n characters and that each character c 2 C is an object with an attribute c:*freq* giving its frequency. The algorithm builds the tree T corresponding to the optimal code in a bottom-up manner. It begins with a set of |C| leaves and performs a sequence of |C| - 1 “merging” operations to create the final tree. The algorithm uses a min-priority queue Q, keyed on the *freq* attribute, to identify the two least-frequent objects to merge together. When we merge two objects, the result is a new object whose frequency is the sum of the frequencies of the two objects that were merged.

HUFFMAN(C)

1 n = |C|

2 Q = C

3 **for** i = 1 **to** n - 1

4 allocate a new node z

5 z.*left =* x = EXTRACT-MIN(Q)

6 z.*right =* y = EXTRACT-MIN(Q)

7 z.*freq =* x.*freq +* y.*freq*

8 INSERT(Q , z)

9 **return** EXTRACT-MIN(Q) **//** return the root of the tree

For our example, Huffman’s algorithm proceeds as shown in Figure 16.5. Since the alphabet contains 6 letters, the initial queue size is n D 6, and 5 merge steps build the tree. The final tree represents the optimal prefix code. The code word for a letter is the sequence of edge labels on the simple path from the root to the letter.

**6.1 VARIABLE LENGTH ENCODING**

Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

**Enter the text:**

A pangram, or holoalphabetic sentence, is a sentence that contains every letter of the alphabet at least once. The most famous pangram is probably the thirty-five-letter-long “The quick brown fox jumps over the lazy dog,” which has been used to test typing equipment since at least the late eighteenth centuary.

**6.1.1 The Input Texts are:**

A pangram, or holoalphabetic sentence, is a sentence that contains every letter of the alphabet at least once. The most famous pangram is probably the thirty-five-letter-long “The quick brown fox jumps over the lazy dog,” which has been used to test typing equipment since at least the late eighteenth centuary.

**Frequency of each character is:**

” 1

“ 1

49

- 3

, 3

. 2

A 1

T 2

a 21

b 6

c 9

d 2

e 37

f 4

g 6

h 16

i 12

j 1

k 1

l 11

m 6

n 17

o 15

p 8

q 2

r 11

s 14

t 31

u 6

v 3

w 2

x 1

y 6

z 1

**(b) Variable Length Binary Code For Each Character:**

{

” 10010000

“ 10010001

' ' 110

- 1001011

, 1111010

. 0001001

A 10010010

T 1000000

a 1010

b 111000

c 10001

d 1000001

e 011

f 000101

g 111001

h 0100

i 10111

j 10010100

k 10010011

l 10011

m 111010

n 0101

o 0000

p 00011

q 1000010

r 10110

s 11111

T 001

u 111011

v 1111011

w 1000011

X 10010101

Y 111100

Z 0001000

}

**6.1.2 Compressed Encoded codes for the entire input text:**

100100101100001110100101111001101101010111010111101011000001011011001000000100110000101010011000110100101011100001100110111100011101111101101010010110101100010111111010110101111111111010101101111101101010010110101100010111100010100101000111010001000001010011010101110101111111100111111011011101101111001101001101100100101110110110000000010111000101000111101010100110001101001010111000011001110101000111010011011101011111001110000001011000101100010011101000000010001111011101000001111100111000010110101110100000111011111111100001110100101111001101101010111010110101111111111000011101100000111000101011100010011111100110001010001111000101001011110110001111100100101100010110111111101101110010111001101100100101110110100101110011000001011110011101001000110000000100011110100001011101110111100011001001111011100010110000010000110101110000101000010010101110100101001110111110100001111111110000011110110111011011000101000111101001110100001000111100110100000100001110011111010100100001101000011010010111100010100110010010101111111011100001101101011101110111111101110000011100010000110001011111110011100011111000001110111010111100111001110000101110111011100011111010011010100111011111101110101100010111101010001110100110111010111110011100010100011110100111010001011110011101111110010100001011011010100101001101000101101010011110111010101101111000001001

**6.1.3 Decoded result :**

A pangram, or holoalphabetic sentence, is a sentence that contains every letter of the alphabet at least once. The most famous pangram is probably the thirty-five-letter-long “The quick brown fox jumps over the lazy dog,” which has been used to test typing equipment since at least the late eighteenth centuary.

>>>

**6.2 FIXED LENGTH ENCODING**

Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

**Enter the text:**

A pangram, or holoalphabetic sentence, is a sentence that contains every letter of the alphabet at least once. The most famous pangram is probably the thirty-five-letter-long “The quick brown fox jumps over the lazy dog,” which has been used to test typing equipment since at least the late eighteenth centuary.

**6.2.1 The Input Texts are:**

A pangram, or holoalphabetic sentence, is a sentence that contains every letter of the alphabet at least once. The most famous pangram is probably the thirty-five-letter-long “The quick brown fox jumps over the lazy dog,” which has been used to test typing equipment since at least the late eighteenth centuary.

**Frequency of each character is:**

” 1

“ 1

49

- 3

, 3

. 2

A 1

T 2

a 21

c 9

b 6

e 37

d 2

g 6

f 4

i 12

h 16

k 1

j 1

m 6

l 11

o 15

n 17

q 2

p 8

s 14

r 11

u 6

t 31

w 2

v 3

y 6

x 1

z 1

**(b)** **The fixed length binary code for each character is:**

a 00000

b 00001

c 00010

d 00011

e 00100

f 00101

g 00110

h 00111

i 01000

j 01001

k 01010

l 01011

m 01100

n 01101

o 01110

p 01111

q 10000

r 10001

s 10010

t 10011

u 10100

v 10101

w 10110

x 10111

y 11000

z 11001

" 11010

- 11011

11100

, 11101

. 11110

**6.2.2 Compressed Encoded codes for the entire input text:**

011110000001101001101000100000011000111010001001110111001011011100000001011011110011100000000010010010011010000001010010001000110110011001000110100010001000100010010000001001000100011011001100100011010001000100100110011100000100110001001110011011001100000010000110110010001001010100100100011100001011001001001110011001001000101110001011001100111001000000001011011110011100000000010010010011000001001101011001000000010010100110111001101000100010000111001000110001110100101001100101000000110001110101001001001111000000110100110100010000001100010001001001111100010111000001000000000101011110001001100111001001001100111010001000110011110000010101000101010010001011001001001110011001001000101011011100110100110001110010010000101000100000010010100000110001011101011001101001010111010111010011010001100011111001001110101010010010001100110011100100010110000011001110000001101110001101011000111010000001000111001110000010010000010010000100011011010010010001000001110011011101001100100100101001110011110000111101000011010011000100100001010001000011110110000100011011001110010010000110100010001000000010011010110010000000100101001110011001110010001011000001001100100001000100000110001111001100100001000110110011001110001000100011011001110100000001000111000

**6.2.3 Decoded result :**

A pangram, or holoalphabetic sentence, is a sentence that contains every letter of the alphabet at least once. The most famous pangram is probably the thirty-five-letter-long “The quick brown fox jumps over the lazy dog,” which has been used to test typing equipment since at least the late eighteenth centuary.

>>>

**6.3 COMPARISIONS**

|  |  |  |
| --- | --- | --- |
| **CHARACTER** | **VARIABLE LENGHT CODING** | **FIXED LENGTH CODING** |
| Left Double Quote( “ ) | 10010001 | 11111 |
| Right Double Quote( ” ) | 10010000 | 11010 |
| Space ( ) | 110 | 11100 |
| Hyphen ( - ) | 1001011 | 11011 |
| Comma ( , ) | 1111010 | 11101 |
| Period ( . ) | 0001001 | 11110 |
| a | 1010 | 00000 |
| b | 111000 | 00001 |
| c | 10001 | 00010 |
| d | 1000001 | 00011 |
| e | 011 | 00100 |
| f | 000101 | 00101 |
| g | 111001 | 00110 |
| h | 0100 | 00111 |
| i | 10111 | 01000 |
| j | 10010100 | 01001 |
| k | 10010011 | 01010 |
| l | 10011 | 01011 |
| m | 111010 | 01100 |
| n | 0101 | 01101 |
| o | 0000 | 01110 |
| p | 00011 | 01111 |
| q | 1000010 | 10000 |
| r | 10110 | 10001 |
| s | 11111 | 10010 |
| t | 001 | 10011 |
| u | 111011 | 10100 |
| v | 1111011 | 10101 |
| w | 1000011 | 10110 |
| x | 10010101 | 10111 |
| y | 111100 | 11000 |
| z | 0001000 | 11001 |

|  |  |  |
| --- | --- | --- |
| **PARAGRAPHS** | **VARIABLE LENGHT CODING** | **FIXED LENGTH CODING** |
| A pangram, or holoalphabetic sentence, is a sentence that containsevery letter of the alphabet at least once. The most famous pangram is probably the thirty-five-letter-long “The quick brown fox jumps over the lazy dog, which has been used to test typing equipment since at least the late eighteenth centuary. | 100100101100001110100101111001101101010111010111101011000001011011001000000100110000101010011000110100101011100001100110111100011101111101101010010110101100010111111010110101111111111010101101111101101010010110101100010111100010100101000111010001000001010011010101110101111111100111111011011101101111001101001101100100101110110110000000010111000101000111101010100110001101001010111000011001110101000111010011011101011111001110000001011000101100010011101000000010001111011101000001111100111000010110101110100000111011111111100001110100101111001101101010111010110101111111111000011101100000111000101011100010011111100110001010001111000101001011110110001111100100101100010110111111101101110010111001101100100101110110100101110011000001011110011101001000110000000100011110100001011101110111100011001001111011100010110000010000110101110000101000010010101110100101001110111110100001111111110000011110110111011011000101000111101001110100001000111100110100000100001110011111010100100001101000011010010111100010100110010010101111111011100001101101011101110111111101110000011100010000110001011111110011100011111000001110111010111100111001110000101110111011100011111010011010100111011111101110101100010111101010001110100110111010111110011100010100011110100111010001011110011101111110010100001011011010100101001101000101101010011110111010101101111000001001 | 011110000001101001101000100000011000111010001001110111001011011100000001011011110011100000000010010010011010000001010010001000110110011001000110100010001000100010010000001001000100011011001100100011010001000100100110011100000100110001001110011011001100000010000110110010001001010100100100011100001011001001001110011001001000101110001011001100111001000000001011011110011100000000010010010011000001001101011001000000010010100110111001101000100010000111001000110001110100101001100101000000110001110101001001001111000000110100110100010000001100010001001001111100010111000001000000000101011110001001100111001001001100111010001000110011110000010101000101010010001011001001001110011001001000101011011100110100110001110010010000101000100000010010100000110001011101011001101001010111010111010011010001100011111001001110101010010010001100110011100100010110000011001110000001101110001101011000111010000001000111001110000010010000010010000100011011010010010001000001110011011101001100100100101001110011110000111101000011010011000100100001010001000011110110000100011011001110010010000110100010001000000010011010110010000000100101001110011001110010001011000001001100100001000100000110001111001100100001000110110011001110001000100011011001110100000001000111000 |
| **NUMBER OF BITS** | **1246** | **1334** |
| Huffman Coding is an entropy encoding algorithm used for lossless data compression. It refers to the use of a variable-length code table for encoding a source symbol, such as character in a file where, the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol. | 11010000001100011100111110001101100001111101011101010100100001000011000011100010110111101100001110100000001001111010100000100001111010000011001101010010000100001100001111011110111100001010011100010010100111100011110011001100101001011100111101001111111101110100110011011011010011001101111001010110010101111111001101011000110000001110100110011000011010000011010101111110101111001011101110100011101001110110111001010101110010100110101110011001100101111010001111111011111110100110110111000110111000111011010110100011101101000001100000010100111111100110101001001011100101011100011101101011100111101001111110100000110011010100100001000011000011110111110110101000110011111001010111011010000111000110001101011011110101001110110001101100110011111101101101111100110011101101111011110010010010011111100010000111101111100111000111011010111110101101001101001110101101010011100101001101011111010011011011100011011100011101101011010001110110100000110000001010011111110011010100100101110010101110001110110101111001110110110111100010100100000111100100100111000111010010101001011100010000111101111110000010110111001000011100100110110111011011111111010110101110000111110001101101100101001011110100000111001010011010111010011000100001110001101100100101001011110000001111010100011011100010001110110001001010000111110100011111110101100111001001100111011101000001100101011100111101001111110101011110011001111110000010100110011000011000111011010111110100110111101100110010111101000111111001010011010111011010100011001111100101011101101000011100011000110101101111010101 | 1010000101001010110000000011010111000011010000110100110010001001000000011010010001101100111000101110011111100000100011010001001110000110100001101001100000001011001100111010001010001001100111011001010010010001000001100101011101000101011011101001010010010110010010010100100001100000100110000000010011100110001111100010010010010100100100001110011011001110001001000010100100100011001010011011101001100111001001010010010001000111000101000001010100000100010100000000000010101100100010110010001101001101001100111000100111000011001001001100000000010101100100001010111010001001000110100010011100001101000011010011000000100100111010100100010001000100100101100001100000010111001011100101010000010001110000010010000100011100000100010000000010100110010010001010000110100000001010100001011001001011000111001001000100100100110011100100101010000010001010000000000001010110010001011001000110100110100110011100010011100001100100100110000000001010110010000111000001001000001001000010001101000110010010001010001010100100000110100001101000000111100000100011001101000000101010001011000001000110110000001100000001000001001000100000110111001101100110011100100001001001010011010000110000000100110010000011011111000101110000010000000001010000101101000100111100001110001010111000010000101010010001100010010001101000100010000101011101000100100000000001000111011110111010010100100100000001010110010010101000000101110100001000111000101100110011100100100100111010100100010001000100100101100001100000010111001011 |
| **NUMBER OF BITS** | **1481** | **1544** |
| A priority queue is an abstract data type which is like a regular queue or some other data structures, but where additionally each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority. If two elements have the same priority, they are served according to their order in the queue. | 1101111101111101001100101100111100101100001010111110110110010011100100111111011101011111000010111111000101000010101000110010001010010001110100010000001000111000010101101000111111101010010101110100100101111011101011110100110111101111110111111000111110001110100011001001001100011001111101101100100111001001111100111100111101010011100110011111001100000100111100111010001000000100011110101000110010010101001000100101100011101011001110111101000010010000111110101001001111000111111000010000100010110001011001101011100001001010010101011101110001010010010111011010010111001100110101100011100101000101011111000111110111001101001100101100111100101100001010110111001111000101011010100111010011011100000001101000111110101101100000101111011000100111111111011101010111111000111110100110010110011110010110000101011111011011001001110010011100111011110000101111101101001011100110011010110001111101011011000001011100101011101000100101111101001100101100111100101100001010111101110101111101010111100110110001101000111101000001111011110001111000111111000010111110110100101110011001101011000111110101101100000101110100100111101011111101001100101100111100101100001010100111111111011101110111101110001101010011111011010010111001100110101100010101111001010001101100011111000001001111110101100010011001111111010011001011001111001011000010101001110111000001001101010111100011000111111010101111001101100011010001111000101001101001001111000100010110101110100011110000011111000001001110111100111001111000100001111001111011010111110000010011111110110110010011100100111001111 | 011111000101000011101000101000100111100010000101000010010100001000100010010000000110100000000011001010011100010000000010100110001100000100110000010011110000111100100101100011101000000100011101000100100101101000010100010000000100010010000110101000101100000100011000010100001001010000100011101000110010011100110000100011101001100111001001000100011000001001100000100101001110001101000001010011101001000100100100100000110100100111011000111001001000100100000000001100011010001001101000011100110100000010110101111000001000000000010001110010001011001000110000100011011001100111000001001000000011111000101000011101000101000100111100000000100101001001110000100100000000100110010000011101100100010011001110100010011011010000001111100010100001110100010100010011110001000010100001001010000100000000110100100010110010001100001000110110011101100100010011001110011101000001100011101111100010100001110100010100010011110000100010010100100010010001101010010000011000010010000101011101000100100000000110100100010110010001100001000110110011101100100010011001110101101110101100111110001010000111010001010001001111000001011001110110011100010001011001000110000100011011001110010001110000010101001001001100111001001001000000011000010001111100010100001110100010100010011110001001100111001001100000000100010010010010001001000110101001000001100000000100001001110100010001101000011010011010011011101001100111001000100010001011101000100011001001000101000011011001100111001001000010100001001010000100 |
| **NUMBER OF BITS** | **1471** | **1543** |

**PARAGRAPH 1 :**

Total number of bits by fixed length coding : 1334

Total number of bits by variable length coding : 1246, **Bits saved : 1334-1246 = 88**

**PARAGRAPH 2 :**

Total number of bits by fixed length coding : 1544

Total number of bits by variable length coding : 1481, **Bits saved : 1544-1481 = 63**

**PARAGRAPH 3 :**

Total number of bits by fixed length coding : 1543

Total number of bits by variable length coding : 1471, **Bits saved : 1543-1471 = 72**

**6.4 SOURCE PROGRAMS**

Implemented in **Python** programming language.

**6.4.1 VARIABLE LENGTH ENCODING**

import operator

import itertools

class NodeType:

INTERNAL = 0

LEAF = 1

class Node:

def \_\_init\_\_(self, node\_type):

self.node\_type = node\_type

self.char = ‘ ‘

self.value = 0

self.left = None

self.right = None

def \_\_le\_\_(self, node):

if self.value <= node.value:

return True

else:

return False

def \_\_repr\_\_(self):

return self.\_\_str\_\_()

def \_\_str\_\_(self):

if self.left != None:

print (self.left)

if self.right != None:

print (self.right)

return self.char + ":" + str(self.value)

def \_\_contains\_\_(self, ch):

return self.char == ch

def update\_char(self, ch):

self.char = ch

def update\_value(self, value):

self.value = value

def traversal(root, sequence, coding):

if root == None:

return

if root.node\_type == NodeType.LEAF:

if len(sequence) == 0:

sequence.append('0')

coding[root.char] = sequence

else:

if root.left != None:

s = sequence[:]

s.append('0')

traversal(root.left, s, coding)

if root.right != None:

s = sequence[:]

s.append('1')

traversal(root.right, s, coding)

def huffman\_encode(src, coding):

# get value of each char

value = {}

for char in src:

if char in value:

value[char] += 1

else:

value[char] = 1

value = (sorted(value.items(), key=operator.itemgetter(1)))

# get nodes

root = []

for key, v in value:

node = Node(NodeType.LEAF)

node.update\_char(key)

node.update\_value(v)

root.append(node)

# generate huffman tree, root is root[0]

while len(root) != 1:

node = Node(NodeType.INTERNAL)

node.update\_value(root[0].value + root[1].value)

node.left = root[0]

node.right = root[1]

root.pop(0)

root.pop(0)

if len(root) == 0:

root.append(node)

else:

for i in range(0, len(root)):

if node.value < root[i].value:

root.insert(i, node)

break

else:

root.append(node)

# traversal the tree to get encoding

sequence = []

traversal(root[0], sequence, coding)

result = []

for char in src:

result.extend(coding[char])

return result

def huffman\_decode(src, coding):

# generate a huffman tree by coding table

root = Node(NodeType.INTERNAL)

for k in coding:

temp = root

for item in coding[k]:

if item == '0':

if temp.left == None:

temp.left = Node(NodeType.INTERNAL)

temp = temp.left

if item == '1':

if temp.right == None:

temp.right = Node(NodeType.INTERNAL)

temp = temp.right

temp.node\_type = NodeType.LEAF

temp.char = k

# decoding by tree

result = []

temp = root

for c in src:

if c == '0':

if temp.left == None:

raise Exception("Unexpected token")

temp = temp.left

if c == '1':

if temp.left == None:

raise Exception("Unexpected token")

temp = temp.right

if temp.node\_type == NodeType.LEAF:

result.append(temp.char)

temp = root

return result

if \_\_name\_\_ == "\_\_main\_\_":

s = input("Enter the text: ")

print("\n (a) The Input Texts are: ")

print(s)

coding = {}

result = huffman\_encode(s, coding)

print ("\n Dictionary ")

print (''.join(coding))

dicti = ''.join(coding)

print("\n Frequency of each character is: ")

for char in dicti:

count = s.count(char)

if count >= 1:

print (char, “\t”, count)

c = []

c = coding

print ("\n (b) Variable Length Binary Code For Each Character: ")

print (c)

print ("\n Compressed Encoded codes for the entire input text: ")

print (''.join(result))

result = huffman\_decode(result, coding)

print ("\n (c) Decoded result : ")

print (''.join(result))

**6.4.2 FIXED LENGTH ENCODING**

import operator

import itertools

class NodeType:

INTERNAL = 0

LEAF = 1

class Node:

def \_\_init\_\_(self, node\_type):

self.node\_type = node\_type

self.char = ‘ ‘

self.value = 0

self.left = None

self.right = None

def \_\_le\_\_(self, node):

if self.value <= node.value:

return True

else:

return False

def \_\_repr\_\_(self):

return self.\_\_str\_\_()

def \_\_str\_\_(self):

if self.left != None:

print (self.left)

if self.right != None:

print (self.right)

return self.char + ":" + str(self.value)

def \_\_contains\_\_(self, ch):

return self.char == ch

def update\_char(self, ch):

self.char = ch

def update\_value(self, value):

self.value = value

def traversal(root, sequence, coding):

if root == None:

return

if root.node\_type == NodeType.LEAF:

if len(sequence) == 0:

sequence.append('0')

coding[root.char] = sequence

else:

if root.left != None:

s = sequence[:]

s.append('0')

traversal(root.left, s, coding)

if root.right != None:

s = sequence[:]

s.append('1')

traversal(root.right, s, coding)

def huffman\_fix\_encode(txt):

n = 5

code = ' '

l1 = ['"','-',' ',',','.']

l = [bin(x)[2:].rjust(n, '0') for x in range(2\*\*n)]

a= list(map(chr, range(97, 123)))

t = a+l1

print("\n",t)

print("\n (b) The fixed length binary code for each character is: ")

for k in range(0,31):

print("\n",t[k],"\t",l[k])

for char in txt:

if char in a:

i = a.index(char)

code = code+l[i]

print("\n Compressed Encoded codes for the entire input text: ")

print("\n",code)

def huffman\_encode(src, coding):

# get value of each char

value = {}

for char in src:

if char in value:

value[char] += 1

else:

value[char] = 1

value = (sorted(value.items(), key=operator.itemgetter(1)))

# get nodes

root = []

for key, v in value:

node = Node(NodeType.LEAF)

node.update\_char(key)

node.update\_value(v)

root.append(node)

# generate huffman tree, root is root[0]

while len(root) != 1:

node = Node(NodeType.INTERNAL)

node.update\_value(root[0].value + root[1].value)

node.left = root[0]

node.right = root[1]

root.pop(0)

root.pop(0)

if len(root) == 0:

root.append(node)

else:

for i in range(0, len(root)):

if node.value < root[i].value:

root.insert(i, node)

break

else:

root.append(node)

# traversal the tree to get encoding

sequence = []

traversal(root[0], sequence, coding)

result = []

for char in src:

result.extend(coding[char])

return result

def huffman\_decode(src, coding):

# generate a huffman tree by coding table

root = Node(NodeType.INTERNAL)

for k in coding:

temp = root

for item in coding[k]:

if item == '0':

if temp.left == None:

temp.left = Node(NodeType.INTERNAL)

temp = temp.left

if item == '1':

if temp.right == None:

temp.right = Node(NodeType.INTERNAL)

temp = temp.right

temp.node\_type = NodeType.LEAF

temp.char = k

# decoding by tree

result = []

temp = root

for c in src:

if c == '0':

if temp.left == None:

raise Exception("Unexpected token")

temp = temp.left

if c == '1':

if temp.left == None:

raise Exception("Unexpected token")

temp = temp.right

if temp.node\_type == NodeType.LEAF:

result.append(temp.char)

temp = root

return result

if \_\_name\_\_ == "\_\_main\_\_":

s = input("Enter the text: ")

print("\n (a) The Input Texts are: ")

print(s)

coding = {}

result = huffman\_encode(s, coding)

dicti = ''.join(coding)

print("\n Frequency of each character is: ")

for char in dicti:

count = s.count(char)

if count >= 1:

print (char, “\t”, count)

huffman\_fix\_encode(s)

result = huffman\_decode(result, coding)

print ("\n (c) Decoded result : ")

print (''.join(result))

**CHAPTER 7**

**PRIORITY QUEUE PROBLEM USING HEAPIFY OR HEAP SORT**

**7.1 PROBLEM DESCRIPTION AND EXPLANATION**

A ***priority queue*** is a data structure for maintaining a set S of elements, each with an associated value called a ***key***. A ***max-priority queue*** supports the following operations:

INSERT(S, x) inserts the element x into the set S, which is equivalent to the operation S = S U {x}.

MAXIMUM(S) returns the element of S with the largest key.

EXTRACT-MAX(S) removes and returns the element of S with the largest key.

INCREASE-KEY(S, x, k) increases the value of element x’s key to the new value k, which is assumed to be at least as large as x’s current key value.

Among their other applications, we can use max-priority queues to schedule jobs on a shared computer. The max-priority queue keeps track of the jobs to be performed and their relative priorities. When a job is finished or interrupted, the scheduler selects the highest-priority job from among those pending by calling EXTRACT-MAX. The scheduler can add a new job to the queue at any time by calling INSERT.

The procedure HEAP-MAXIMUM implements the MAXIMUM operation in Ɵ(1) time.

HEAP-MAXIMUM(A)  
1 **return** A[1]

The procedure HEAP-EXTRACT-MAX implements the EXTRACT-MAX operation. It is similar to the **for** loop body (lines 3–5) of the HEAPSORT procedure. The running time of HEAP-EXTRACT-MAX is O(lg n).

HEAP-EXTRACT-MAX(A)

1 **if** A.*heap*-*size* < 1

2 **error** “heap underflow”

3 *max =* A[1]

4 A[1] = A[A.*heap*-*size]*

5 A.*heap*-*size =* A.*heap*-*size -* 1

6 MAX-HEAPIFY(A, 1)

7 **return** *max*

The procedure HEAP-INCREASE-KEY implements the INCREASE-KEY operation. An index i into the array identifies the priority-queue element whose key we wish to increase. The procedure first updates the key of element A[i] to its new value. Because increasing the key of A[i] might violate the max-heap property, the procedure then, traverses a simple path from this node toward the root to find a proper place for the newly increased key. As HEAP-INCREASEKEY traverses this path, it repeatedly compares an element to its parent, exchanging their keys and continuing if the element’s key is larger, and terminating if the element’s key is smaller, since the max-heap property now holds.

HEAP-INCREASE-KEY(A, i, *key)*

1 **if** *key* < A[i]

2 **error** “new key is smaller than current key”

3 A[i] = *key*

4 **while** i > 1 and A[PARENT(i )] < A[i]

5 exchange A[i] with A[PARENT(i )]

6 i = PARENT(i )

The procedure MAX-HEAP-INSERT implements the INSERT operation. It takes as an input the key of the new element to be inserted into max-heap A. The procedure first expands the max-heap by adding to the tree a new leaf whose key is 1. Then it calls HEAP-INCREASE-KEY to set the key of this new node to its correct value and maintain the max-heap property.

MAX-HEAP-INSERT(A, *key)*

1 A.*heap*-*size =* A.*heap*-*size +* 1

2 A[A.*heap*-*size]* = 1

3 HEAP-INCREASE-KEY(A,A.*heap*-*size.* *key)*

The running time of MAX-HEAP-INSERT on an n-element heap is O(lg n).

**7.2 TABULATION, OUTPUT AND ANALYSIS**

Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

**Enter number of jobs:** 12

Enter the job number: J1

Enter the job name: JOB1

Enter submitter: A

Enter Priority: 7

Enter the job number: J2

Enter the job name: JOB2

Enter submitter: B

Enter Priority: 2

Enter the job number: J3

Enter the job name: JOB3

Enter submitter: C

Enter Priority: 1

Enter the job number: J4

Enter the job name: JOB4

Enter submitter: D

Enter Priority: 3

Enter the job number: J5

Enter the job name: JOB5

Enter submitter: E

Enter Priority: 4

Enter the job number: J6

Enter the job name: JOB6

Enter submitter: F

Enter Priority: 6

Enter the job number: J7

Enter the job name: JOB7

Enter submitter: G

Enter Priority: 5

Enter the job number: J8

Enter the job name: JOB8

Enter submitter: H

Enter Priority: 10

Enter the job number: J9

Enter the job name: JOB9

Enter submitter: I

Enter Priority: 8

Enter the job number: J10

Enter the job name: JOB10

Enter submitter: J

Enter Priority: 11

Enter the job number: J11

Enter the job name: JOB11

Enter submitter: K

Enter Priority: 9

Enter the job number: J12

Enter the job name: JOB12

Enter submitter: L

Enter Priority: 12

**7.2.1 WAIT QUEUE**

Job Number: J12

Job Name: JOB12

Submitter Name: L

Priority: 12

Job Number: J10

Job Name: JOB10

Submitter Name: J

Priority: 11

Job Number: J8

Job Name: JOB8

Submitter Name: H

Priority: 10

Job Number: J6

Job Name: JOB6

Submitter Name: F

Priority: 6

Job Number: J1

Job Name: JOB1

Submitter Name: A

Priority: 7

Job Number: J11

Job Name: JOB11

Submitter Name: K

Priority: 9

Job Number: J2

Job Name: JOB2

Submitter Name: B

Priority: 2

Job Number: J7

Job Name: JOB7

Submitter Name: G

Priority: 5

Job Number: J3

Job Name: JOB3

Submitter Name: C

Priority: 1

Job Number: J5

Job Name: JOB5

Submitter Name: E

Priority: 4

Job Number: J4

Job Name: JOB4

Submitter Name: D

Priority: 3

Job Number: J9

Job Name: JOB9

Submitter Name: I

Priority: 8

**Moving highest priority job from wait queue to ready queue:**

**Moving highest priority job from wait queue to ready queue 2nd time:**

**Moving highest priority job from wait queue to ready queue 3rd time:**

**7.2.2 Result after 3 moves from wait queue to ready queue:**

**WAIT QUEUE**

Job Number: J11

Job Name: JOB11

Submitter Name: K

Priority: 9

Job Number: J9

Job Name: JOB9

Submitter Name: I

Priority: 8

Job Number: J1

Job Name: JOB1

Submitter Name: A

Priority: 7

Job Number: J6

Job Name: JOB6

Submitter Name: F

Priority: 6

Job Number: J5

Job Name: JOB5

Submitter Name: E

Priority: 4

Job Number: J4

Job Name: JOB4

Submitter Name: D

Priority: 3

Job Number: J2

Job Name: JOB2

Submitter Name: B

Priority: 2

Job Number: J7

Job Name: JOB7

Submitter Name: G

Priority: 5

Job Number: J3

Job Name: JOB3

Submitter Name: C

Priority: 1

**READY QUEUE**

Job Number: J12

Job Name: JOB12

Submitter Name: L

Priority: 12

Job Number: J10

Job Name: JOB10

Submitter Name: J

Priority: 11

Job Number: J8

Job Name: JOB8

Submitter Name: H

Priority: 10

**7.2.3 Ready Queue after removing two jobs is:**

Job Number: J8

Job Name: JOB8

Submitter Name: H

Priority: 10

**Enter number of jobs: 2**

Enter the job number: J13

Enter the job name: JOB13

Enter submitter: M

Enter Priority: 14

Enter the job number: J14

Enter the job name: JOB14

Enter submitter: N

Enter Priority: 13

**WAIT QUEUE**

Job Number: J13

Job Name: JOB13

Submitter Name: M

Priority: 14

Job Number: J14

Job Name: JOB14

Submitter Name: N

Priority: 13

Job Number: J11

Job Name: JOB11

Submitter Name: K

Priority: 9

Job Number: J6

Job Name: JOB6

Submitter Name: F

Priority: 6

Job Number: J1

Job Name: JOB1

Submitter Name: A

Priority: 7

Job Number: J9

Job Name: JOB9

Submitter Name: I

Priority: 8

Job Number: J2

Job Name: JOB2

Submitter Name: B

Priority: 2

Job Number: J7

Job Name: JOB7

Submitter Name: G

Priority: 5

Job Number: J3

Job Name: JOB3

Submitter Name: C

Priority: 1

Job Number: J5

Job Name: JOB5

Submitter Name: E

Priority: 4

Job Number: J4

Job Name: JOB4

Submitter Name: D

Priority: 3

**Moving highest priority job from wait queue to ready queue:**

**Moving highest priority job from wait queue to ready queue 2nd time:**

**7.2.4 Adding two jobs to WAIT QUEUE, moving two jobs to READY QUUEUE and removing one job from READY QUEUE**

**Result:**

**WAIT QUEUE**

Job Number: J11

Job Name: JOB11

Submitter Name: K

Priority: 9

Job Number: J9

Job Name: JOB9

Submitter Name: I

Priority: 8

Job Number: J1

Job Name: JOB1

Submitter Name: A

Priority: 7

Job Number: J6

Job Name: JOB6

Submitter Name: F

Priority: 6

Job Number: J5

Job Name: JOB5

Submitter Name: E

Priority: 4

Job Number: J4

Job Name: JOB4

Submitter Name: D

Priority: 3

Job Number: J2

Job Name: JOB2

Submitter Name: B

Priority: 2

Job Number: J7

Job Name: JOB7

Submitter Name: G

Priority: 5

Job Number: J3

Job Name: JOB3

Submitter Name: C

Priority: 1

**READY QUEUE**

Job Number: J14

Job Name: JOB14

Submitter Name: N

Priority: 13

Job Number: J8

Job Name: JOB8

Submitter Name: H

Priority: 10

**The Jobs in the wait queue are:**

JOB11

JOB9

JOB1

JOB6

JOB5

JOB4

JOB2

JOB7

JOB3

**Enter the job name for which you want to change the priority:** JOB1

**Enter new priority value:** 15

**Enter the job name for which you want to change the priority:** JOB4

**Enter new priority value:** 16

**7.2.5 Result after changing priority of two jobs:**

**WAIT QUEUE**

Job Number: J4

Job Name: JOB4

Submitter Name: D

Priority: 16

Job Number: J1

Job Name: JOB1

Submitter Name: A

Priority: 15

Job Number: J11

Job Name: JOB11

Submitter Name: K

Priority: 9

Job Number: J9

Job Name: JOB9

Submitter Name: I

Priority: 8

Job Number: J6

Job Name: JOB6

Submitter Name: F

Priority: 6

Job Number: J7

Job Name: JOB7

Submitter Name: G

Priority: 5

Job Number: J3

Job Name: JOB3

Submitter Name: C

Priority: 1

Job Number: J5

Job Name: JOB5

Submitter Name: E

Priority: 4

Job Number: J4

Job Name: JOB4

Submitter Name: D

Priority: 3

**7.2.6 Doing more**

**f. Removing 2 jobs from READY QUEUE and moving 3 jobs from WAIT QUEUE to READY QUEUE:**

**The result is:**

**WAIT QUEUE:**

Job Number: J9

Job Name: JOB9

Submitter Name: I

Priority: 8

Job Number: J6

Job Name: JOB6

Submitter Name: F

Priority: 6

Job Number: J7

Job Name: JOB7

Submitter Name: G

Priority: 5

Job Number: J5

Job Name: JOB5

Submitter Name: E

Priority: 4

Job Number: J4

Job Name: JOB4

Submitter Name: D

Priority: 3

Job Number: J3

Job Name: JOB3

Submitter Name: C

Priority: 1

**READY QUEUE:**

Job Number: J4

Job Name: JOB4

Submitter Name: D

Priority: 16

Job Number: J1

Job Name: JOB1

Submitter Name: A

Priority: 15

Job Number: J11

Job Name: JOB11

Submitter Name: K

Priority: 9

**Removing one job from READY QUEUE**

**Result:**

**WAIT QUEUE:**

Job Number: J9

Job Name: JOB9

Submitter Name: I

Priority: 8

Job Number: J6

Job Name: JOB6

Submitter Name: F

Priority: 6

Job Number: J7

Job Name: JOB7

Submitter Name: G

Priority: 5

Job Number: J5

Job Name: JOB5

Submitter Name: E

Priority: 4

Job Number: J4

Job Name: JOB4

Submitter Name: D

Priority: 3

Job Number: J3

Job Name: JOB3

Submitter Name: C

Priority: 1

**READY QUEUE:**

Job Number: J1

Job Name: JOB1

Submitter Name: A

Priority: 15

Job Number: J11

Job Name: JOB11

Submitter Name: K

Priority: 9

**Moving 5 jobs from WAIT QUEUE to READY QUEUE:**

**The result is:**

**WAIT QUEUE:**

Job Number: J3

Job Name: JOB3

Submitter Name: C

Priority: 1

**READY QUEUE:**

Job Number: J1

Job Name: JOB1

Submitter Name: A

Priority: 15

Job Number: J11

Job Name: JOB11

Submitter Name: K

Priority: 9

Job Number: J9

Job Name: JOB9

Submitter Name: I

Priority: 8

Job Number: J6

Job Name: JOB6

Submitter Name: F

Priority: 6

Job Number: J7

Job Name: JOB7

Submitter Name: G

Priority: 5

Job Number: J5

Job Name: JOB5

Submitter Name: E

Priority: 4

Job Number: J4

Job Name: JOB4

Submitter Name: D

Priority: 3

**Removing 4 jobs from READY QUEUE**

**Result:**

**WAIT QUEUE:**

Job Number: J3

Job Name: JOB3

Submitter Name: C

Priority: 1

**READY QUEUE:**

Job Number: J7

Job Name: JOB7

Submitter Name: G

Priority: 5

Job Number: J5

Job Name: JOB5

Submitter Name: E

Priority: 4

Job Number: J4

Job Name: JOB4

Submitter Name: D

Priority: 3

**Removing 2 more jobs from READY QUEUE**

**Result:**

**WAIT QUEUE:**

Job Number: J3

Job Name: JOB3

Submitter Name: C

Priority: 1

**READY QUEUE:**

Job Number: J4

Job Name: JOB4

Submitter Name: D

Priority: 3

**Moving remaining jobs from WAIT QUEUE to READY QUEUE:**

**The result is:**

**WAIT QUEUE: []**

**READY QUEUE:**

Job Number: J4

Job Name: JOB4

Submitter Name: D

Priority: 3

Job Number: J3

Job Name: JOB3

Submitter Name: C

Priority: 1

**Removing remaining jobs from READY QUEUE:**

**The result is:**

**WAIT QUEUE: []**

**READY QUEUE: []**

>>>

**7.3 SOURCE PROGRAM**

Implemented in **Python** programming language.

DISTANCE\_INDEX = 1

import math

class PriorityQueue:

""" This class illustrates a PriorityQueue and its associated functions """

def \_\_init\_\_(self, k):

self.heap = []

self.k = k

def items(self):

return self.heap

def parent(self, index):

"""

Parent will be at math.floor(index/2). Since integer division

simulates the floor function, we don't explicity use it

"""

return math.floor(index/2)

def left\_child(self, index):

return 2 \* index

def right\_child(self, index):

return (2 \* index) + 1

def max\_heapify(self, index):

"""

Responsible for maintaining the heap property of the heap.

This function assumes that the subtree located at left and right

child satisfies the max-heap property. But the tree at index

(current node) does not. O(log n)

"""

left\_index = self.left\_child(index)

right\_index = self.right\_child(index)

largest = index

if left\_index < len(self.heap) and self.heap[left\_index][DISTANCE\_INDEX] > self.heap[index][DISTANCE\_INDEX]:

largest = left\_index

if right\_index < len(self.heap) and self.heap[right\_index][DISTANCE\_INDEX] > self.heap[largest][DISTANCE\_INDEX]:

largest = right\_index

if largest != index:

self.heap[index], self.heap[largest] = self.heap[largest], self.heap[index]

self.max\_heapify(largest)

def build\_max\_heap(self):

"""

Responsible for building the heap bottom up. It starts with the lowest non-leaf nodes

and calls heapify on them. This function is useful for initialising a heap with an

unordered array. O(n)

We shall note that all the elements after floor(size/2) are leaves.

"""

for i in range(len(self.heap)):

self.max\_heapify(i)

def heap\_sort(self):

""" The heap-sort algorithm with a time complexity O(n\*log(n))

We run n times the max\_heapify (O(log n))

"""

self.build\_max\_heap()

output = []

for i in range(len(self.heap)-1, 0, -1):

self.heap[0], self.heap[i] = self.heap[i], self.heap[0]

output.append(self.heap.pop())

self.max\_heapify(0)

output.append(self.heap.pop())

self.heap = output

def propagate\_up(self, index):

""" Compares index with parent and swaps node if larger O(log(n)) """

while index != 0 and self.heap[self.parent(index)][DISTANCE\_INDEX] < self.heap[index][DISTANCE\_INDEX]:

self.heap[index], self.heap[self.parent(index)] = self.heap[self.parent(index)], self.heap[index]

index = self.parent(index)

# Here is the whole logic of the Priority queue.

# Add an element only if size < k and if size == k, only if the element value is less than

def add(self, obj):

# If number of elements == k and new element < max\_elem:

# extract\_max and add the new element.

# Else:

# Add the new element.

size = self.size()

# Size == k, The priority queue is at capacity.

if size == self.k:

max\_elem = self.max()

# The new element has a lower distance than the biggest one.

# Then we insert, otherwise, don't insert.

if obj[DISTANCE\_INDEX] < max\_elem:

self.extract\_max()

self.heap\_append(obj)

# if size == 0 or 0 < Size < k

else:

self.heap\_append(obj)

def heap\_append(self, obj):

""" Adds an element in the heap O(ln(n)) """

self.heap.append(obj)

self.propagate\_up(len(self.heap) - 1) # Index value is 1 less than length

def max(self):

# The highest distance will always be at the index 0 (heap invariant)

return self.heap[0][1]

def size(self):

return len(self.heap)

def extract\_max(self):

"""

Part of the Priority Queue, extracts the element on the top of the heap and

then re-heapifies. O(log n)

"""

max = self.heap[0]

data = self.heap.pop()

if len(self.heap) > 0:

self.heap[0] = data

self.max\_heapify(0)

return max

def increment(self, key, value):

""" Increments key by the input value. O(log n) """

for i in range(len(self.heap)):

if self.heap[i][0] == key:

self.heap[i][0] = (key, (value + self.heap[i][1]))

self.propagate\_up(i)

break

if \_\_name\_\_ == '\_\_main\_\_':

# Create the heap object

m = PriorityQueue(15)

ready = []

ind = []

ind1 = []

j1 = []

j = []

s = []

p = []

p1 = []

p2 = []

u1 = int(input("\n Enter number of jobs: "))

for b in range(u1):

u2 = input("\n Enter the job number: ")

u3 = input("\n Enter the job name: ")

u4 = input("\n Enter submitter: ")

u5 = int(input("\n Enter Priority: "))

j.append(u2)

j1.append(u3)

s.append(u4)

p.append(u5)

# Add elements to the heap

for i in range(u1):

m.add((j1[i],p[i]))

def Display(data, m, n):

for i in range(len(data)):

m.append(data[i][0])

if m[i] in j1:

z = j1.index(m[i])

n.append(z)

y = 0

for x in n:

print("\n\n Job Number: ",j[x])

print("\n Job Name: ",data[y][0])

print("\n Submitter Name: ",s[x])

print("\n Priority: ",data[y][1])

y = y+1

print("\n\n\n a. WAIT QUEUE ")

Display(m.heap, p1, ind)

print("\n\n Moving highest priority job from wait queue to ready queue: ")

a1 = m.extract\_max()

ready.append(a1)

print("\n Moving highest priority job from wait queue to ready queue 2nd time: ")

a2 = m.extract\_max()

ready.append(a2)

print("\n Moving highest priority job from wait queue to ready queue 3rd time: ")

a3 = m.extract\_max()

ready.append(a3)

print("\n\n b. Result after 3 moves from wait queue to ready queue: ")

print("\n\n WAIT QUEUE ")

Display(m.heap, p2, ind1)

ind1 = []

p2 = []

print("\n\n READY QUEUE ")

Display(ready, p2, ind1)

ind1 = []

p2 = []

ready.pop(0)

ready.pop(0)

print("\n c. Ready Queue after removing two jobs is: ")

Display(ready, p2, ind1)

v1 = int(input("\n Enter number of jobs: "))

for b in range(v1):

v2 = input("\n Enter the job number: ")

v3 = input("\n Enter the job name: ")

v4 = input("\n Enter submitter: ")

v5 = int(input("\n Enter Priority: "))

j.append(v2)

j1.append(v3)

s.append(v4)

p.append(v5)

for i in range(v1):

m.add((j1[u1+i],p[u1+i]))

ind1 = []

p2 = []

print("\n\n WAIT QUEUE ")

Display(m.heap,p2,ind1)

print("\n\n Moving highest priority job from wait queue to ready queue: ")

a1 = m.extract\_max()

ready.append(a1)

print("\n Moving highest priority job from wait queue to ready queue 2nd time: ")

a2 = m.extract\_max()

ready.append(a2)

a5 = ready.pop(0)

ready.append(a5)

ready.pop(0)

print("\n d. Adding two jobs to WAIT QUEUE, moving two jobs to READY QUUEUE and removing one job from READY QUEUE ")

print("\n Result: ")

ind1 = []

p2 = []

print("\n\n WAIT QUEUE ")

Display(m.heap,p2,ind1)

ind1 = []

p2 = []

print("\n\n READY QUEUE ")

Display(ready,p2,ind1)

g = []

for l in range(len(m.heap)):

g1 = m.heap[l][0]

g.append(g1)

print("\n\n The Jobs in the wait queue are: ")

for l in range(len(g)):

print("\n",g[l])

w1 = str(input("\n Enter the job name for which you want to change the priority: "))

p3 = int(input("\n Enter new priority value: "))

w2 = str(input("\n Enter the job name for which you want to change the priority: "))

p4 = int(input("\n Enter new priority value: "))

p.append(p3)

p.append(p4)

for l2 in range(len(m.heap)):

if m.heap[l2][0] == w1:

i1 = l2

if m.heap[l2][0] == w2:

i2 = l2

m.heap.pop(i1)

m.heap.pop(i2)

m.add((w1,p3))

m.add((w2,p4))

ind1 = []

p2 = []

print("\n\n e. Result after changing priority of two jobs: ")

print("\n WAIT QUEUE ")

Display(m.heap,p2,ind1)

print("\n\n f. Removing 2 jobs from READY QUEUE and moving 3 jobs from WAIT QUEUE to READY QUEUE: ")

ready.pop(0)

ready.pop(0)

ready.append(m.extract\_max())

ready.append(m.extract\_max())

ready.append(m.extract\_max())

print("\n The result is: ")

print("\n WAIT QUEUE: ")

ind1 = []

p2 = []

Display(m.heap,p2,ind1)

print("\n READY QUEUE: ")

ind1 = []

p2 = []

Display(ready,p2,ind1)

print("\n Removing one job from READY QUEUE ")

print("\n Result: ")

ready.pop(0)

print("\n WAIT QUEUE: ")

ind1 = []

p2 = []

Display(m.heap,p2,ind1)

print("\n READY QUEUE: ")

ind1 = []

p2 = []

Display(ready,p2,ind1)

print("\n\n Moving 5 jobs from WAIT QUEUE to READY QUEUE: ")

ready.append(m.extract\_max())

ready.append(m.extract\_max())

ready.append(m.extract\_max())

ready.append(m.extract\_max())

ready.append(m.extract\_max())

print("\n The result is: ")

print("\n WAIT QUEUE: ")

ind1 = []

p2 = []

Display(m.heap,p2,ind1)

print("\n READY QUEUE: ")

ind1 = []

p2 = []

Display(ready,p2,ind1)

print("\n Removing 4 jobs from READY QUEUE ")

ready.pop(0)

ready.pop(0)

ready.pop(0)

ready.pop(0)

print("\n Result: ")

print("\n WAIT QUEUE: ")

ind1 = []

p2 = []

Display(m.heap,p2,ind1)

print("\n READY QUEUE: ")

ind1 = []

p2 = []

Display(ready,p2,ind1)

print("\n Removing 2 more jobs from READY QUEUE ")

ready.pop(0)

ready.pop(0)

print("\n Result: ")

print("\n WAIT QUEUE: ")

ind1 = []

p2 = []

Display(m.heap,p2,ind1)

print("\n READY QUEUE: ")

ind1 = []

p2 = []

Display(ready,p2,ind1)

print("\n\n Moving remaining jobs from WAIT QUEUE to READY QUEUE: ")

ready.append(m.extract\_max())

print("\n The result is: ")

print("\n WAIT QUEUE: ",m.heap)

print("\n READY QUEUE: ")

ind1 = []

p2 = []

Display(ready,p2,ind1)

print("\n\n Removing remaining jobs from READY QUEUE: ")

ready.pop(0)

ready.pop(0)

print("\n The result is: ")

print("\n WAIT QUEUE: ",m.heap)

print("\n READY QUEUE: ",ready)

**CHAPTER 8**

**MATRICES CHAIN MULTIPLICATION PROBLEM AND DYNAMIC PROGRAMMING**

**8 PROBLEM DESCRIPTION**

**Matrix chain multiplication** (or Matrix Chain Ordering Problem, MCOP) is an optimization problem that can be solved using [dynamic programming](https://en.wikipedia.org/wiki/Dynamic_programming). Given a sequence of matrices, the goal is to find the most efficient way to [multiply these matrices](https://en.wikipedia.org/wiki/Matrix_multiplication). The problem is not actually to *perform* the multiplications, but merely to decide the sequence of the matrix multiplications involved.

We have many options because matrix multiplication is [associative](https://en.wikipedia.org/wiki/Associativity). In other words, no matter how we [parenthesize](https://en.wikipedia.org/wiki/Bracket_(mathematics)) the product, the result obtained will remain the same. For example, if we had four matrices *A*, *B*, *C*, and *D*, we would have:

((*AB*)*C*)*D* = ((*A*(*BC*))*D*) = (*AB*)(*CD*) = *A*((*BC*)*D*) = *A*(*B*(*CD*)).

However, the order in which we parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the *efficiency*. For example, suppose A is a 10 × 30 matrix, B is a 30 × 5 matrix, and C is a 5 × 60 matrix. Then,

(*AB*)*C* = (10×30×5) + (10×5×60) = 1500 + 3000 = 4500 operations

*A*(*BC*) = (30×5×60) + (10×30×60) = 9000 + 18000 = 27000 operations.

Clearly the first method is more efficient. With this information, the problem statement can be refined, how do we determine the optimal parenthesization of a product of *n*matrices? We could go through each possible parenthesization ([brute force](https://en.wikipedia.org/wiki/Brute-force_search)), requiring a [run-time](https://en.wikipedia.org/wiki/Time_complexity) that is exponential in the number of matrices, which is very slow and impractical for large *n*. A quicker solution to this problem can be achieved by breaking up the problem into a set of related subproblems. By solving subproblems one time and reusing these solutions, we can drastically reduce the run-time required. This concept is known as [dynamic programming](https://en.wikipedia.org/wiki/Dynamic_programming). To begin, let us assume that all we really want to know is the minimum cost, or minimum number of arithmetic operations, needed to multiply out the matrices. If we are only multiplying two matrices, there is only one way to multiply them, so the minimum cost is the cost of doing this. In general, we can find the minimum cost using the following [recursive algorithm](https://en.wikipedia.org/wiki/Recursion):

* Take the sequence of matrices and separate it into two subsequences.
* Find the minimum cost of multiplying out each subsequence.
* Add these costs together, and add in the cost of multiplying the two result matrices.
* Do this for each possible position at which the sequence of matrices can be split, and take the minimum over all of them.

Pseudocode:

*// Matrix Ai has dimension p[i-1] x p[i] for i = 1..n*MatrixChainOrder(int p[])  
{  
 *// length[p] = n + 1* n = p.length - 1;  
 *// m[i,j] = Minimum number of scalar multiplications (i.e., cost)* *// needed to compute the matrix A[i]A[i+1]...A[j] = A[i..j]* *// cost is zero when multiplying one matrix* **for** (i = 1; i <= n; i++)

m[i,i] = 0;

**for** (L=2; L<=n; L++) { *// L is chain length* **for** (i=1; i<=n-L+1; i++) {  
 j = i+L-1;  
 m[i,j] = MAXINT;  
 **for** (k=i; k<=j-1; k++) {  
 *// q = cost/scalar multiplications* q = m[i,k] + m[k+1,j] + p[i-1]\*p[k]\*p[j];

**if** (q < m[i,j]) {

m[i,j] = q;

s[i,j]=k; *// s[i,j] = Second auxiliary table that stores k* *// k = Index that achieved optimal cost*

}  
 }  
 }  
 }  
}

**8.1 m[i,j] and s[i,j] TABULATION WITH SAME MATRICES AS IN THE TEXTBOOK**

**SET 1 :**

Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

Enter total number of matrices: 6

Enter the matrix dimension: 30

Enter the matrix dimension: 35

Enter the matrix dimension: 15

Enter the matrix dimension: 5

Enter the matrix dimension: 10

Enter the matrix dimension: 20

Enter the matrix dimension: 25

**(a) The entered dimensions are:**

[30, 35, 15, 5, 10, 20, 25]

**The most efficient way to multiply all the matrices are:**

(

(

A[ 1 ]

(

A[ 2 ]

A[ 3 ]

)

)

(

(

A[ 4 ]

A[ 5 ]

)

A[ 6 ]

)

)

**(a) UPPER TRIANGULAR MATRIX m[i,j]:**

0 15750 7875 9375 11875 15125

0 2625 4375 7125 10500

0 750 2500 5375

0 1000 3500

0 5000

0

**(a) UPPER TRIANGULAR MATRIX s[i,j]:**

0 1 1 3 3 3

0 2 3 3 3

0 3 3 3

0 4 5

0 5

0

**(a) Total scalar multiplication of the matrix chain using dynamic programming technology:**

15125

**(b) Total scalar multiplication of the matrix chain without using dynamic programming technology:**

40500

>>>

**8.2 m[i,j] and s[i,j] TABULATIONS OF OTHER MATRICES**

Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

**SET 2 :**

Enter total number of matrices: 7

Enter the matrix dimension: 25

Enter the matrix dimension: 5

Enter the matrix dimension: 20

Enter the matrix dimension: 10

Enter the matrix dimension: 15

Enter the matrix dimension: 35

Enter the matrix dimension: 30

Enter the matrix dimension: 10

**(a) The entered dimensions are:**

[25, 5, 20, 10, 15, 35, 30, 10]

**The most efficient way to multiply all the matrices are:**

(

A[ 1 ]

(

(

(

(

(

A[ 2 ]

A[ 3 ]

)

A[ 4 ]

)

A[ 5 ]

)

A[ 6 ]

)

A[ 7 ]

)

)

**(a) UPPER TRIANGULAR MATRIX m[i,j]:**

0 2500 2250 3625 8750 13375 12375

0 1000 1750 4375 9625 11125

0 3000 12250 21750 19250

0 5250 15750 17250

0 15750 15750

0 10500

0

**(a) UPPER TRIANGULAR MATRIX s[i,j]:**

0 1 1 1 1 1 1

0 2 3 4 5 6

0 3 3 3 3

0 4 5 4

0 5 5

0 6

0

**(a) Total scalar multiplication of the matrix chain using dynamic programming technology:**

12375

**(b) Total scalar multiplication of the matrix chain without using dynamic programming technology:**

58125

>>>

**SET 3 :**

Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

Enter total number of matrices: 8

Enter the matrix dimension: 5

Enter the matrix dimension: 20

Enter the matrix dimension: 35

Enter the matrix dimension: 15

Enter the matrix dimension: 25

Enter the matrix dimension: 10

Enter the matrix dimension: 40

Enter the matrix dimension: 5

Enter the matrix dimension: 15

**(a) The entered dimensions are:**

[5, 20, 35, 15, 25, 10, 40, 5, 15]

**The most efficient way to multiply all the matrices are:**

(

(

(

(

(

(

A[ 1 ]

A[ 2 ]

)

A[ 3 ]

)

A[ 4 ]

)

A[ 5 ]

)

(

A[ 6 ]

A[ 7 ]

)

)

A[ 8 ]

)

**(a) UPPER TRIANGULAR MATRIX m[i,j]:**

0 3500 6125 8000 9250 11250 11500 11875

0 10500 18000 16000 24000 11250 12750

0 13125 9000 23000 7750 10375

0 3750 9750 5125 6250

0 10000 3250 5125

0 2000 2750

0 3000

0

**(a) UPPER TRIANGULAR MATRIX s[i,j]:**

0 1 2 3 4 5 5 7

0 2 3 2 5 2 7

0 3 3 5 3 7

0 4 5 4 7

0 5 5 7

0 6 7

0 7

0

**(a) Total scalar multiplication of the matrix chain using dynamic programming technology:**

11875

**(b) Total scalar multiplication of the matrix chain without using dynamic programming technology:**

12625

>>>

**SET 4**

Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

Enter total number of matrices: 9

Enter the matrix dimension: 10

Enter the matrix dimension: 5

Enter the matrix dimension: 25

Enter the matrix dimension: 40

Enter the matrix dimension: 30

Enter the matrix dimension: 15

Enter the matrix dimension: 20

Enter the matrix dimension: 10

Enter the matrix dimension: 35

Enter the matrix dimension: 15

**(a) The entered dimensions are:**

[10, 5, 25, 40, 30, 15, 20, 10, 35, 15]

**The most efficient way to multiply all the matrices are:**

(

A[ 1 ]

(

(

(

(

(

(

(

A[ 2 ]

A[ 3 ]

)

A[ 4 ]

)

A[ 5 ]

)

A[ 6 ]

)

A[ 7 ]

)

A[ 8 ]

)

A[ 9 ]

)

)

**(a) UPPER TRIANGULAR MATRIX m[i,j]:**

0 1250 7000 12500 14000 15750 16250 19250 20875

0 5000 11000 13250 14750 15750 17500 20125

0 30000 33000 40500 29500 38250 38500

0 18000 30000 19500 33500 30750

0 9000 7500 18000 17250

0 3000 8250 10500

0 7000 8250

0 5250

0

**(a) UPPER TRIANGULAR MATRIX s[i,j]:**

0 1 1 1 1 1 1 1 1

0 2 3 4 5 6 7 8

0 3 3 5 3 7 7

0 4 5 4 7 7

0 5 5 7 5

0 6 7 7

0 7 7

0 8

0

**(a) Total scalar multiplication of the matrix chain using dynamic programming technology:**

20875

**(b) Total scalar multiplication of the matrix chain without using dynamic programming technology:**

41500

>>>

**SET 5 :**

Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

Enter total number of matrices: 10

Enter the matrix dimension: 35

Enter the matrix dimension: 10

Enter the matrix dimension: 45

Enter the matrix dimension: 5

Enter the matrix dimension: 20

Enter the matrix dimension: 15

Enter the matrix dimension: 25

Enter the matrix dimension: 40

Enter the matrix dimension: 15

Enter the matrix dimension: 35

Enter the matrix dimension: 20

**(a) The entered dimensions are:**

[35, 10, 45, 5, 20, 15, 25, 40, 15, 35, 20]

**The most efficient way to multiply all the matrices are:**

(

(

A[ 1 ]

(

A[ 2 ]

A[ 3 ]

)

)

(

(

(

(

(

(

A[ 4 ]

A[ 5 ]

)

A[ 6 ]

)

A[ 7 ]

)

A[ 8 ]

)

A[ 9 ]

)

A[ 10 ]

)

)

**(a) UPPER TRIANGULAR MATRIX m[i,j]:**

0 15750 4000 7500 8125 11750 19375 18000 24125 25000

0 2250 3250 4500 6875 12625 14375 18000 20750

0 4500 4875 9000 17375 14750 21875 22000

0 1500 3375 8375 11375 14000 17500

0 7500 27000 25125 35625 41625

0 15000 20625 28500 35625

0 15000 28125 33000

0 21000 22500

0 10500

0

**(a) UPPER TRIANGULAR MATRIX s[i,j]:**

0 1 1 3 3 3 3 3 3 3

0 2 3 3 3 3 3 3 3

0 3 3 3 3 3 3 3

0 4 5 6 7 8 9

0 5 5 5 8 5

0 6 6 8 8

0 7 8 8

0 8 8

0 9

0

**(a) Total scalar multiplication of the matrix chain using dynamic programming technology:**

25000

**(b) Total scalar multiplication of the matrix chain without using dynamic programming technology:**

149625

**COMPARISION OF (a) AND (b)**

|  |  |  |  |
| --- | --- | --- | --- |
| **SET** | **(a)** | **(b)** | **(b) - (a)** |
| 1 | 15125 | 40500 | 25375 |
| 2 | 12375 | 58125 | 45750 |
| 3 | 11875 | 12625 | 750 |
| 4 | 20875 | 41500 | 20625 |
| 5 | 25000 | 149625 | 124625 |

**SET 1**

(a) Total scalar multiplication of the matrix chain using dynamic programming technology:

15125

(b) Total scalar multiplication of the matrix chain without using dynamic programming technology:

40500

**Number of multiplications saved = 40500 - 15125 = 25375**

**SET 2**

(a) Total scalar multiplication of the matrix chain using dynamic programming technology:

12375

(b) Total scalar multiplication of the matrix chain without using dynamic programming technology:

58125

**Number of multiplications saved = 58125 - 12375 = 45750**

**SET 3**

(a) Total scalar multiplication of the matrix chain using dynamic programming technology:

11875

(b) Total scalar multiplication of the matrix chain without using dynamic programming technology:

12625

**Number of multiplications saved = 12625 - 11875 = 750**

**SET 4**

(a) Total scalar multiplication of the matrix chain using dynamic programming technology:

20875

(b) Total scalar multiplication of the matrix chain without using dynamic programming technology:

41500

**Number of multiplications saved = 41500 - 20875 = 20625**

**SET 5**

(a) Total scalar multiplication of the matrix chain using dynamic programming technology:

25000

(b) Total scalar multiplication of the matrix chain without using dynamic programming technology:

149625

**Number of multiplications saved = 149625 - 25000 = 124625**

**8.4 SOURCE PROGRAM**

Implemented in **Python** programming language.

import os

p = []

z = int(input("\n Enter total number of matrices: "))

for y in range(z+1):

r = int(input("\n Enter the matrix dimension: "))

p.append(r)

print("\n (a) The entered dimensions are: ")

print (p)

def print\_parens(s, i, j ):

if( i==j ) :

print ("A[", i, "]",)

else :

print (' (')

#print ( '( ', end=""),

print\_parens(s, i, s[i][j] )

print\_parens(s, s[i][j]+1, j)

print (') ')

#print ( ') ', end=''),

def main():

n=( len(p)-1 )

m=[[0 for x in range(n+1)] for x in range(n+1)]

s=[[0 for x in range(n+1)] for x in range(n+1)]

for l in range(2,n+1) : #l=chain length

for i in range(1,n-l+2) :

j=i+l-1

m[i][j]=999999999 #infinity

for k in range(i,j):

q=m[i][k] + m[k+1][j] + p[i-1]\*p[k]\*p[j]

if(q<m[i][j]) :

m[i][j]=q

s[i][j]=k

#Now lets print the output with parentheses

print("\n The most efficient way to multiply all the matrices are: ")

print\_parens(s,1, n)

m.pop(0)

m.pop(len(m)-1)

print("\n\n (a) UPPER TRIANGULAR MATRIX m[i,j]: ")

y=0

for i in range(len(m)):

m[i].pop(0)

print("\n",m[i])

s.pop(0)

s.pop(len(s)-1)

print("\n\n (a) UPPER TRIANGULAR MATRIX s[i,j]: ")

for j in range(len(s)):

s[j].pop(0)

print("\n",s[j])

c = len(m[0])-1

print("\n Total scalar multiplication of the matrix chain using dynamic programming technology: \n",m[0][c])

cnt = 0

for b in range(len(p)-2):

cnt = cnt + (p[0]\*p[b+1]\*p[b+2])

print("\n (b) Total scalar multiplication of the matrix chain without using dynamic programming technology: \n", cnt)

main()

**CHAPTER 9**

**BINARY SEARCH TREE PROBLEM**

**9.1 PROBLEM DESCRIPTION AND EXPLANATION**

A binary search tree is a [rooted](https://en.wikipedia.org/wiki/Rooted_tree) [binary tree](https://en.wikipedia.org/wiki/Binary_tree), whose internal nodes each store a key (and optionally, an associated value) and each have two distinguished sub-trees, commonly denoted *left* and *right*. The tree additionally satisfies the binary search tree property, which states that the key in each node must be greater than all keys stored in the left sub-tree, and smaller than all keys in right sub-tree.[[1]](https://en.wikipedia.org/wiki/Binary_search_tree#cite_note-1) (The leaves (final nodes) of the tree contain no key and have no structure to distinguish them from one another. Leaves are commonly represented by a special leaf or nil symbol, a NULL pointer, etc.)

Generally, the information represented by each node is a record rather than a single data element. However, for sequencing purposes, nodes are compared according to their keys rather than any part of their associated records.

The major advantage of binary search trees over other data structures is that the related [sorting algorithms](https://en.wikipedia.org/wiki/Sorting_algorithm) and [search algorithms](https://en.wikipedia.org/wiki/Search_algorithm) such as [in-order traversal](https://en.wikipedia.org/wiki/In-order_traversal) can be very efficient; they are also easy to code.

Binary search trees are a fundamental data structure used to construct more abstract data structures such as [sets](https://en.wikipedia.org/wiki/Set_(computer_science)), [multisets](https://en.wikipedia.org/wiki/Set_(computer_science)#Multiset), and [associative arrays](https://en.wikipedia.org/wiki/Associative_array).

**Insertion**To insert a new value into a binary search tree T , we use the procedure TREEINSERT. The procedure takes a node ’ for which ’:*key* D , ’:*left* D NIL, and ’:*right* D NIL. It modifies T and some of the attributes of ’ in such a way that it inserts ’ into an appropriate position in the tree.

TREE-INSERT(T, z)  
1 y = NIL  
2 x = T.*root*3 **while** x ≠ NIL  
4 y = x  
5 **if** z.*key* < x.*key*6 x = x.*left*7 **else** x = x.*right*8 z.*p =* y  
9 **if** y == NIL  
10 T.*root =* z **//** tree T was empty  
11 **elseif** z**.***key* < y.*key*12 y.*left =* z  
13 **else** y.*right =* z

**Searching**We use the following procedure to search for a node with a given key in a binary search tree. Given a pointer to the root of the tree and a key k, TREE-SEARCH returns a pointer to a node with key k if one exists; otherwise, it returns NIL.

TREE-SEARCH(x, k)  
1 if x == NIL or k == x.key  
2 return x  
3 if k < x.key  
4 return TREE-SEARCH(x.left, k)  
5 else return TREE-SEARCH(x.right, k)

Deletion  
In order to move subtrees around within the binary search tree, we define a subroutine TRANSPLANT, which replaces one subtree as a child of its parent with another subtree. When TRANSPLANT replaces the subtree rooted at node u with the subtree rooted at node v, node u’s parent becomes node v’s parent, and u’s parent ends up having v as its appropriate child.

TRANSPLANT(T, u, v)  
1 if u.p == NIL  
2 T.root = v  
3 elseif u == u.p.left  
4 u.p.left = v  
5 else u.p.right = v  
6 if v ≠ NIL  
7 v.p = u.p

With the TRANSPLANT procedure in hand, here is the procedure that deletes node z from binary search tree

TREE-DELETE(T, z)  
1 if z.left == NIL  
2 TRANSPLANT(T, z, z.right)  
3 elseif z.right == NIL  
4 TRANSPLANT(T, z, z.left)  
5 else y = TREE-MINIMUM(z.right)  
6 if y.p ≠ z  
7 TRANSPLANT(T, y, y.right)  
8 y.right = z.right  
9 y.right.p = y  
10 TRANSPLANT(T, z, y)  
11 y.left = z.left  
12 y.left.p = y

**9.2 TABULATION, OUTPUT AND ANALYSIS**

**SET 1 :**

Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

**Enter total number of students:** 22

**Enter Student ID:** 11

**Enter Student Name:** STUDENT K

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 16

**Enter Student Name:** STUDENT P

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: C

Enter Grade: A

Enter Grade: D

**Enter Student ID:** 15

**Enter Student Name:** STUDENT O

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: D

Enter Grade: B

Enter Grade: B

Enter Grade: F

Enter Grade: A

**Enter Student ID:** 9

**Enter Student Name:** STUDENT I

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: D

**Enter Student ID:** 18

**Enter Student Name:** STUDENT R

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

**Enter Student ID:** 12

**Enter Student Name:** STUDENT L

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: A

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 2

**Enter Student Name:** STUDENT B

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: A

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 10

**Enter Student Name:** STUDENT J

Enter Grade: C

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: D

Enter Grade: B

Enter Grade: A

Enter Grade: B

Enter Grade: C

Enter Grade: A

**Enter Student ID:** 1

**Enter Student Name:** STUDENT A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: A

Enter Grade: B

**Enter Student ID:** 8

**Enter Student Name:** STUDENT H

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: A

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 21

**Enter Student Name:** STUDENT U

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 14

**Enter Student Name:** STUDENT N

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

**Enter Student ID:** 0

**Enter Student Name:** STUDENT W

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 3

**Enter Student Name:** STUDENT C

Enter Grade: A

Enter Grade: B

Enter Grade: A

Enter Grade: B

Enter Grade: A

Enter Grade: B

Enter Grade: C

Enter Grade: A

Enter Grade: B

Enter Grade: D

**Enter Student ID:** 19

**Enter Student Name:** STUDENT S

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 17

**Enter Student Name:** STUDENT Q

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: C

Enter Grade: A

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 7

**Enter Student Name:** STUDENT G

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: D

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: A

**Enter Student ID:** 6

**Enter Student Name:** STUDENT F

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

**Enter Student ID:** 13

**Enter Student Name:** STUDENT M

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: A

Enter Grade: D

**Enter Student ID:** 4

**Enter Student Name:** STUDENT D

Enter Grade: C

Enter Grade: C

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: A

Enter Grade: D

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 20

**Enter Student Name:** STUDENT T

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: D

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 5

**Enter Student Name:** STUDENT E

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: A

Enter Grade: B

Enter Grade: A

**9.2.1 Unsorted List of Student Records:**

[[11, 'STUDENT K', 'p'], [16, 'STUDENT P', 'p'], [15, 'STUDENT O', 'p'], [9, 'STUDENT I', 'p'], [18, 'STUDENT R', 'p'], [12, 'STUDENT L', 'p'], [2, 'STUDENT B', 'p'], [10, 'STUDENT J', 'p'], [1, 'STUDENT A', 'p'], [8, 'STUDENT H', 'p'], [21, 'STUDENT U', 'p'], [14, 'STUDENT N', 'p'], [0, 'STUDENT W', 'p'], [3, 'STUDENT C', 'p'], [19, 'STUDENT S', 'p'], [17, 'STUDENT Q', 'p'], [7, 'STUDENT G', 'p'], [6, 'STUDENT F', 'p'], [13, 'STUDENT M', 'p'], [4, 'STUDENT D', 'p'], [20, 'STUDENT T', 'p'], [5, 'STUDENT E', 'p']]

**9.2.2 In order Traversal of the Binary Search Tree:**

Student ID: 0

Student Name: STUDENT W

Grades: ['A', 'B', 'A', 'B', 'C', 'B', 'B', 'A', 'D', 'A']

Student ID: 1

Student Name: STUDENT A

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'A']

Student ID: 2

Student Name: STUDENT B

Grades: ['A', 'A', 'B', 'B', 'B', 'B', 'A', 'C', 'A', 'D']

Student ID: 3

Student Name: STUDENT C

Grades: ['A', 'A', 'A', 'A', 'C', 'D', 'B', 'B', 'F', 'A']

Student ID: 4

Student Name: STUDENT D

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'D']

Student ID: 5

Student Name: STUDENT E

Grades: ['A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A']

Student ID: 6

Student Name: STUDENT F

Grades: ['B', 'B', 'A', 'A', 'C', 'C', 'C', 'A', 'B', 'A']

Student ID: 7

Student Name: STUDENT G

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'C', 'A', 'B', 'A']

Student ID: 8

Student Name: STUDENT H

Grades: ['C', 'A', 'A', 'C', 'D', 'B', 'A', 'B', 'C', 'A']

Student ID: 9

Student Name: STUDENT I

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'A', 'B']

Student ID: 10

Student Name: STUDENT J

Grades: ['A', 'A', 'B', 'B', 'C', 'C', 'C', 'A', 'B', 'A']

Student ID: 11

Student Name: STUDENT K

Grades: ['B', 'B', 'B', 'A', 'A', 'A', 'A', 'C', 'B', 'A']

Student ID: 12

Student Name: STUDENT L

Grades: ['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']

Student ID: 13

Student Name: STUDENT M

Grades: ['A', 'A', 'A', 'C', 'C', 'C', 'B', 'B', 'B', 'A']

Student ID: 14

Student Name: STUDENT N

Grades: ['A', 'B', 'A', 'B', 'A', 'B', 'C', 'A', 'B', 'D']

Student ID: 15

Student Name: STUDENT O

Grades: ['A', 'A', 'C', 'C', 'C', 'B', 'B', 'B', 'B', 'A']

Student ID: 16

Student Name: STUDENT P

Grades: ['A', 'A', 'B', 'B', 'B', 'C', 'C', 'A', 'B', 'A']

Student ID: 17

Student Name: STUDENT Q

Grades: ['B', 'B', 'B', 'B', 'D', 'A', 'A', 'A', 'C', 'A']

Student ID: 18

Student Name: STUDENT R

Grades: ['C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C']

Student ID: 19

Student Name: STUDENT S

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'A', 'D']

Student ID: 20

Student Name: STUDENT T

Grades: ['C', 'C', 'A', 'A', 'A', 'B', 'A', 'D', 'B', 'A']

Student ID: 21

Student Name: STUDENT U

Grades: ['C', 'C', 'C', 'D', 'A', 'A', 'A', 'A', 'B', 'A']

**9.2.3 Searching two ID’s**

**Enter First ID to be searched:** 7

**Enter Second ID to be searched:** 19

**First ID Found:**

Student ID: 7

Student Name: STUDENT G

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'C', 'A', 'B', 'A']

**Second ID Found:**

Student ID: 19

Student Name: STUDENT S

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'A', 'D']

**Enter ID to be deleted:** 18

**Enter ID to be deleted:** 2

**9.2.4 In order Traversal of the Binary Search Tree After Deleting two nodes:**

Student ID: 0

Student Name: STUDENT W

Grades: ['A', 'B', 'A', 'B', 'C', 'B', 'B', 'A', 'D', 'A']

Student ID: 1

Student Name: STUDENT A

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'A']

Student ID: 3

Student Name: STUDENT C

Grades: ['A', 'A', 'A', 'A', 'C', 'D', 'B', 'B', 'F', 'A']

Student ID: 4

Student Name: STUDENT D

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'D']

Student ID: 5

Student Name: STUDENT E

Grades: ['A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A']

Student ID: 6

Student Name: STUDENT F

Grades: ['B', 'B', 'A', 'A', 'C', 'C', 'C', 'A', 'B', 'A']

Student ID: 7

Student Name: STUDENT G

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'C', 'A', 'B', 'A']

Student ID: 8

Student Name: STUDENT H

Grades: ['C', 'A', 'A', 'C', 'D', 'B', 'A', 'B', 'C', 'A']

Student ID: 9

Student Name: STUDENT I

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'A', 'B']

Student ID: 10

Student Name: STUDENT J

Grades: ['A', 'A', 'B', 'B', 'C', 'C', 'C', 'A', 'B', 'A']

Student ID: 11

Student Name: STUDENT K

Grades: ['B', 'B', 'B', 'A', 'A', 'A', 'A', 'C', 'B', 'A']

Student ID: 12

Student Name: STUDENT L

Grades: ['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']

Student ID: 13

Student Name: STUDENT M

Grades: ['A', 'A', 'A', 'C', 'C', 'C', 'B', 'B', 'B', 'A']

Student ID: 14

Student Name: STUDENT N

Grades: ['A', 'B', 'A', 'B', 'A', 'B', 'C', 'A', 'B', 'D']

Student ID: 15

Student Name: STUDENT O

Grades: ['A', 'A', 'C', 'C', 'C', 'B', 'B', 'B', 'B', 'A']

Student ID: 16

Student Name: STUDENT P

Grades: ['A', 'A', 'B', 'B', 'B', 'C', 'C', 'A', 'B', 'A']

Student ID: 17

Student Name: STUDENT Q

Grades: ['B', 'B', 'B', 'B', 'D', 'A', 'A', 'A', 'C', 'A']

Student ID: 19

Student Name: STUDENT S

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'A', 'D']

Student ID: 20

Student Name: STUDENT T

Grades: ['C', 'C', 'A', 'A', 'A', 'B', 'A', 'D', 'B', 'A']

Student ID: 21

Student Name: STUDENT U

Grades: ['C', 'C', 'C', 'D', 'A', 'A', 'A', 'A', 'B', 'A']

**Enter Student ID:** 2

**Enter Student Name:** STUDENT Y

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: C

Enter Grade: A

Enter Grade: D

**9.2.5 In order Traversal of the Binary Search Tree after Inserting one node:**

Student ID: 0

Student Name: STUDENT W

Grades: ['A', 'B', 'A', 'B', 'C', 'B', 'B', 'A', 'D', 'A']

Student ID: 1

Student Name: STUDENT A

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'B', 'A', 'B', 'A']

Student ID: 2

Student Name: STUDENT Y

Grades: ['A', 'A', 'B', 'B', 'B', 'B', 'A', 'C', 'A', 'D']

Student ID: 3

Student Name: STUDENT C

Grades: ['A', 'A', 'A', 'A', 'C', 'D', 'B', 'B', 'F', 'A']

Student ID: 4

Student Name: STUDENT D

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'D']

Student ID: 5

Student Name: STUDENT E

Grades: ['A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A']

Student ID: 6

Student Name: STUDENT F

Grades: ['B', 'B', 'A', 'A', 'C', 'C', 'C', 'A', 'B', 'A']

Student ID: 7

Student Name: STUDENT G

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'C', 'A', 'B', 'A']

Student ID: 8

Student Name: STUDENT H

Grades: ['C', 'A', 'A', 'C', 'D', 'B', 'A', 'B', 'C', 'A']

Student ID: 9

Student Name: STUDENT I

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'A', 'B']

Student ID: 10

Student Name: STUDENT J

Grades: ['A', 'A', 'B', 'B', 'C', 'C', 'C', 'A', 'B', 'A']

Student ID: 11

Student Name: STUDENT K

Grades: ['B', 'B', 'B', 'A', 'A', 'A', 'A', 'C', 'B', 'A']

Student ID: 12

Student Name: STUDENT L

Grades: ['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']

Student ID: 13

Student Name: STUDENT M

Grades: ['A', 'A', 'A', 'C', 'C', 'C', 'B', 'B', 'B', 'A']

Student ID: 14

Student Name: STUDENT N

Grades: ['A', 'B', 'A', 'B', 'A', 'B', 'C', 'A', 'B', 'D']

Student ID: 15

Student Name: STUDENT O

Grades: ['A', 'A', 'C', 'C', 'C', 'B', 'B', 'B', 'B', 'A']

Student ID: 16

Student Name: STUDENT P

Grades: ['A', 'A', 'B', 'B', 'B', 'C', 'C', 'A', 'B', 'A']

Student ID: 17

Student Name: STUDENT Q

Grades: ['B', 'B', 'B', 'B', 'D', 'A', 'A', 'A', 'C', 'A']

Student ID: 19

Student Name: STUDENT S

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'A', 'D']

Student ID: 20

Student Name: STUDENT T

Grades: ['C', 'C', 'A', 'A', 'A', 'B', 'A', 'D', 'B', 'A']

Student ID: 21

Student Name: STUDENT U

Grades: ['C', 'C', 'C', 'D', 'A', 'A', 'A', 'A', 'B', 'A']

**9.2.6 With More Set’s**

**SET 2 :**

Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

**Enter total number of students:** 25

**Enter Student ID:** 13

**Enter Student Name:** STUDENT M

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: C

Enter Grade: A

**Enter Student ID:** 19

**Enter Student Name:** STUDENT S

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: B

Enter Grade: A

Enter Grade: B

**Enter Student ID:** 10

**Enter Student Name:** STUDENT J

Enter Grade: C

Enter Grade: D

Enter Grade: D

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

**Enter Student ID:** 16

**Enter Student Name:** STUDENT P

Enter Grade: C

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: D

**Enter Student ID:** 1

**Enter Student Name:** STUDENT A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

**Enter Student ID:** 5

**Enter Student Name:** STUDENT E

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: A

**Enter Student ID:** 21

**Enter Student Name:** STUDENT U

Enter Grade: C

Enter Grade: C

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: C

**Enter Student ID:** 9

**Enter Student Name:** STUDENT I

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: B

Enter Grade: C

Enter Grade: A

**Enter Student ID:** 24

**Enter Student Name:** STUDENT X

Enter Grade: C

Enter Grade: C

Enter Grade: D

Enter Grade: D

Enter Grade: D

Enter Grade: D

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 22

**Enter Student Name:** STUDENT V

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: C

**Enter Student ID:** 8

**Enter Student Name:** STUDENT H

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: A

Enter Grade: B

**Enter Student ID:** 12

**Enter Student Name:** STUDENT L

Enter Grade: A

Enter Grade: C

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: C

**Enter Student ID:** 18

**Enter Student Name:** STUDENT R

Enter Grade: C

Enter Grade: C

Enter Grade: D

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: D

Enter Grade: A

Enter Grade: C

**Enter Student ID:** 17

**Enter Student Name:** STUDENT Q

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

**Enter Student ID:** 23

**Enter Student Name:** STUDENT W

Enter Grade: A

Enter Grade: B

Enter Grade: C

Enter Grade: A

Enter Grade: B

Enter Grade: C

Enter Grade: A

Enter Grade: B

Enter Grade: C

Enter Grade: A

**Enter Student ID:** 3

**Enter Student Name:** STUDENT C

Enter Grade: C

Enter Grade: C

Enter Grade: D

Enter Grade: D

Enter Grade: D

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 4

**Enter Student Name:** STUDENT D

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade:

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 6

**Enter Student Name:** STUDENT F

Enter Grade: C

Enter Grade: C

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

**Enter Student ID:** 11

**Enter Student Name:** STUDENT K

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: A

**Enter Student ID:** 20

**Enter Student Name:** STUDENT T

Enter Grade: C

Enter Grade: C

Enter Grade: D

Enter Grade: D

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: D

Enter Grade: B

**Enter Student ID:** 14

**Enter Student Name:** STUDENT N

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: B

**Enter Student ID:** 7

**Enter Student Name:** STUDENT G

Enter Grade: C

Enter Grade: A

Enter Grade: B

Enter Grade: D

Enter Grade: A

Enter Grade: B

Enter Grade: D

Enter Grade: C

Enter Grade: A

Enter Grade: B

**Enter Student ID:** 0

**Enter Student Name:** STUDENT Z

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 15

**Enter Student Name:** STUDENT O

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: V

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 2

**Enter Student Name:** STUDENT B

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: C

**1) Unsorted List of Student Records:**

[[13, 'STUDENT M', 'p'], [19, 'STUDENT S', 'p'], [10, 'STUDENT J', 'p'], [16, 'STUDENT P', 'p'], [1, 'STUDENT A', 'p'], [5, 'STUDENT E', 'p'], [21, 'STUDENT U', 'p'], [9, 'STUDENT I', 'p'], [24, 'STUDENT X', 'p'], [22, 'STUDENT V', 'p'], [8, 'STUDENT H', 'p'], [12, 'STUDENT L', 'p'], [18, 'STUDENT R', 'p'], [17, 'STUDENT Q', 'p'], [23, 'STUDENT W', 'p'], [3, 'STUDENT C', 'p'], [4, 'STUDENT D', 'p'], [6, 'STUDENT F', 'p'], [11, 'STUDENT K ', 'p'], [20, 'STUDENT T', 'p'], [14, 'STUDENT N', 'p'], [7, 'STUDENT G', 'p'], [0, 'STUDENT Z', 'p'], [15, 'STUDENT O', 'p'], [2, 'STUDENT B', 'p']]

**2) In order Traversal of the Binary Search Tree:**

Student ID: 0

Student Name: STUDENT Z

Grades: ['A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'C']

Student ID: 1

Student Name: STUDENT A

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'C', 'C', 'A']

Student ID: 2

Student Name: STUDENT B

Grades: ['B', 'B', 'C', 'A', 'A', 'A', 'C', 'B', 'A', 'B']

Student ID: 3

Student Name: STUDENT C

Grades: ['C', 'D', 'D', 'C', 'C', 'C', 'A', 'A', 'A', 'B']

Student ID: 4

Student Name: STUDENT D

Grades: ['C', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'C', 'D']

Student ID: 5

Student Name: STUDENT E

Grades: ['A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A']

Student ID: 6

Student Name: STUDENT F

Grades: ['B', 'B', 'B', 'A', 'A', 'A', 'C', 'C', 'C', 'A']

Student ID: 7

Student Name: STUDENT G

Grades: ['C', 'C', 'A', 'A', 'A', 'A', 'B', 'B', 'A', 'C']

Student ID: 8

Student Name: STUDENT H

Grades: ['A', 'A', 'C', 'C', 'B', 'B', 'A', 'B', 'C', 'A']

Student ID: 9

Student Name: STUDENT I

Grades: ['C', 'C', 'D', 'D', 'D', 'D', 'A', 'A', 'B', 'A']

Student ID: 10

Student Name: STUDENT J

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'C']

Student ID: 11

Student Name: STUDENT K

Grades: ['B', 'B', 'A', 'A', 'A', 'A', 'C', 'C', 'A', 'B']

Student ID: 12

Student Name: STUDENT L

Grades: ['A', 'C', 'B', 'B', 'C', 'A', 'A', 'B', 'B', 'C']

Student ID: 13

Student Name: STUDENT M

Grades: ['C', 'C', 'D', 'A', 'A', 'A', 'B', 'D', 'A', 'C']

Student ID: 14

Student Name: STUDENT N

Grades: ['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']

Student ID: 15

Student Name: STUDENT O

Grades: ['A', 'B', 'C', 'A', 'B', 'C', 'A', 'B', 'C', 'A']

Student ID: 16

Student Name: STUDENT P

Grades: ['C', 'C', 'D', 'D', 'D', 'A', 'A', 'B', 'B', 'A']

Student ID: 17

Student Name: STUDENT Q

Grades: ['A', 'A', 'C', 'C', 'D', 'B', 'B', 'A', 'B', 'A']

Student ID: 18

Student Name: STUDENT R

Grades: ['C', 'C', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B']

Student ID: 19

Student Name: STUDENT S

Grades: ['A', 'A', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'A']

Student ID: 20

Student Name: STUDENT T

Grades: ['C', 'C', 'D', 'D', 'A', 'A', 'A', 'A', 'D', 'B']

Student ID: 21

Student Name: STUDENT U

Grades: ['A', 'A', 'C', 'C', 'C', 'C', 'B', 'B', 'A', 'B']

Student ID: 22

Student Name: STUDENT V

Grades: ['C', 'A', 'B', 'D', 'A', 'B', 'D', 'C', 'A', 'B']

Student ID: 23

Student Name: STUDENT W

Grades: ['A', 'A', 'A', 'C', 'C', 'C', 'B', 'B', 'B', 'A']

Student ID: 24

Student Name: STUDENT X

Grades: ['A', 'A', 'A', 'A', 'A', 'C', 'B', 'B', 'B', 'A']

**Enter First ID to be searched:** 4

**Enter Second ID to be searched:** 22

**3) First ID Found:**

Student ID: 4

Student Name: STUDENT D

Grades: ['C', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'C', 'D']

**3) Second ID Found:**

Student ID: 22

Student Name: STUDENT V

Grades: ['C', 'A', 'B', 'D', 'A', 'B', 'D', 'C', 'A', 'B']

**Enter ID to be deleted:** 21

**Enter ID to be deleted:** 1

**4) In order Traversal of the Binary Search Tree After Deleting two nodes:**

Student ID: 0

Student Name: STUDENT Z

Grades: ['A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'C']

Student ID: 2

Student Name: STUDENT B

Grades: ['B', 'B', 'C', 'A', 'A', 'A', 'C', 'B', 'A', 'B']

Student ID: 3

Student Name: STUDENT C

Grades: ['C', 'D', 'D', 'C', 'C', 'C', 'A', 'A', 'A', 'B']

Student ID: 4

Student Name: STUDENT D

Grades: ['C', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'C', 'D']

Student ID: 5

Student Name: STUDENT E

Grades: ['A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A']

Student ID: 6

Student Name: STUDENT F

Grades: ['B', 'B', 'B', 'A', 'A', 'A', 'C', 'C', 'C', 'A']

Student ID: 7

Student Name: STUDENT G

Grades: ['C', 'C', 'A', 'A', 'A', 'A', 'B', 'B', 'A', 'C']

Student ID: 8

Student Name: STUDENT H

Grades: ['A', 'A', 'C', 'C', 'B', 'B', 'A', 'B', 'C', 'A']

Student ID: 9

Student Name: STUDENT I

Grades: ['C', 'C', 'D', 'D', 'D', 'D', 'A', 'A', 'B', 'A']

Student ID: 10

Student Name: STUDENT J

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'C']

Student ID: 11

Student Name: STUDENT K

Grades: ['B', 'B', 'A', 'A', 'A', 'A', 'C', 'C', 'A', 'B']

Student ID: 12

Student Name: STUDENT L

Grades: ['A', 'C', 'B', 'B', 'C', 'A', 'A', 'B', 'B', 'C']

Student ID: 13

Student Name: STUDENT M

Grades: ['C', 'C', 'D', 'A', 'A', 'A', 'B', 'D', 'A', 'C']

Student ID: 14

Student Name: STUDENT N

Grades: ['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']

Student ID: 15

Student Name: STUDENT O

Grades: ['A', 'B', 'C', 'A', 'B', 'C', 'A', 'B', 'C', 'A']

Student ID: 16

Student Name: STUDENT P

Grades: ['C', 'C', 'D', 'D', 'D', 'A', 'A', 'B', 'B', 'A']

Student ID: 17

Student Name: STUDENT Q

Grades: ['A', 'A', 'C', 'C', 'D', 'B', 'B', 'A', 'B', 'A']

Student ID: 18

Student Name: STUDENT R

Grades: ['C', 'C', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B']

Student ID: 19

Student Name: STUDENT S

Grades: ['A', 'A', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'A']

Student ID: 20

Student Name: STUDENT T

Grades: ['C', 'C', 'D', 'D', 'A', 'A', 'A', 'A', 'D', 'B']

Student ID: 22

Student Name: STUDENT V

Grades: ['C', 'A', 'B', 'D', 'A', 'B', 'D', 'C', 'A', 'B']

Student ID: 23

Student Name: STUDENT W

Grades: ['A', 'A', 'A', 'C', 'C', 'C', 'B', 'B', 'B', 'A']

Student ID: 24

Student Name: STUDENT X

Grades: ['A', 'A', 'A', 'A', 'A', 'C', 'B', 'B', 'B', 'A']

**Enter Student ID:** 25

**Enter Student Name:** STUDENT X1

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: C

Enter Grade: B

Enter Grade: C

**5) In order Traversal of the Binary Search Tree after Inserting one node:**

Student ID: 0

Student Name: STUDENT Z

Grades: ['A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'C']

Student ID: 2

Student Name: STUDENT B

Grades: ['B', 'B', 'C', 'A', 'A', 'A', 'C', 'B', 'A', 'B']

Student ID: 3

Student Name: STUDENT C

Grades: ['C', 'D', 'D', 'C', 'C', 'C', 'A', 'A', 'A', 'B']

Student ID: 4

Student Name: STUDENT D

Grades: ['C', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'C', 'D']

Student ID: 5

Student Name: STUDENT E

Grades: ['A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A']

Student ID: 6

Student Name: STUDENT F

Grades: ['B', 'B', 'B', 'A', 'A', 'A', 'C', 'C', 'C', 'A']

Student ID: 7

Student Name: STUDENT G

Grades: ['C', 'C', 'A', 'A', 'A', 'A', 'B', 'B', 'A', 'C']

Student ID: 8

Student Name: STUDENT H

Grades: ['A', 'A', 'C', 'C', 'B', 'B', 'A', 'B', 'C', 'A']

Student ID: 9

Student Name: STUDENT I

Grades: ['C', 'C', 'D', 'D', 'D', 'D', 'A', 'A', 'B', 'A']

Student ID: 10

Student Name: STUDENT J

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'C']

Student ID: 11

Student Name: STUDENT K

Grades: ['B', 'B', 'A', 'A', 'A', 'A', 'C', 'C', 'A', 'B']

Student ID: 12

Student Name: STUDENT L

Grades: ['A', 'C', 'B', 'B', 'C', 'A', 'A', 'B', 'B', 'C']

Student ID: 13

Student Name: STUDENT M

Grades: ['C', 'C', 'D', 'A', 'A', 'A', 'B', 'D', 'A', 'C']

Student ID: 14

Student Name: STUDENT N

Grades: ['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']

Student ID: 15

Student Name: STUDENT O

Grades: ['A', 'B', 'C', 'A', 'B', 'C', 'A', 'B', 'C', 'A']

Student ID: 16

Student Name: STUDENT P

Grades: ['C', 'C', 'D', 'D', 'D', 'A', 'A', 'B', 'B', 'A']

Student ID: 17

Student Name: STUDENT Q

Grades: ['A', 'A', 'C', 'C', 'D', 'B', 'B', 'A', 'B', 'A']

Student ID: 18

Student Name: STUDENT R

Grades: ['C', 'C', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B']

Student ID: 19

Student Name: STUDENT S

Grades: ['A', 'A', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'A']

Student ID: 20

Student Name: STUDENT T

Grades: ['C', 'C', 'D', 'D', 'A', 'A', 'A', 'A', 'D', 'B']

Student ID: 22

Student Name: STUDENT V

Grades: ['C', 'A', 'B', 'D', 'A', 'B', 'D', 'C', 'A', 'B']

Student ID: 23

Student Name: STUDENT W

Grades: ['A', 'A', 'A', 'C', 'C', 'C', 'B', 'B', 'B', 'A']

Student ID: 24

Student Name: STUDENT X

Grades: ['A', 'A', 'A', 'A', 'A', 'C', 'B', 'B', 'B', 'A']

Student ID: 25

Student Name: STUDENT X1

Grades: ['A', 'B', 'B', 'A', 'B', 'B', 'A', 'C', 'B', 'C']

**SET 3 :**

Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

**Enter total number of students:** 30

**Enter Student ID:** 16

**Enter Student Name:** STUDENT P

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: C

Enter Grade: A

Enter Grade: B

**Enter Student ID:** 25

**Enter Student Name:** STUDENT Y

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: D

Enter Grade: A

**Enter Student ID:** 26

**Enter Student Name:** STUDENT Z1

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: V

Enter Grade: B

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 3

**Enter Student Name:** STUDENT C

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

**Enter Student ID:** 14

**Enter Student Name:** STUDENT N

Enter Grade: B

Enter Grade: B

Enter Grade: D

Enter Grade: A

Enter Grade: A

Enter Grade: D

Enter Grade: C

Enter Grade: A

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 13

**Enter Student Name:** STUDENT M

Enter Grade: C

Enter Grade: C

Enter Grade: D

Enter Grade: D

Enter Grade: D

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: C

**Enter Student ID:** 4

**Enter Student Name:** STUDENT D

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: C

**Enter Student ID:** 29

**Enter Student Name:** STUDENT Y6

Enter Grade: A

Enter Grade: B

Enter Grade: A

Enter Grade: B

Enter Grade: A

Enter Grade: B

Enter Grade: A

Enter Grade: B

Enter Grade: A

Enter Grade: A

**Enter Student ID:** 28

**Enter Student Name:** STUDENT P4

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: A

Enter Grade: C

**Enter Student ID:** 17

**Enter Student Name:** STUDENT Q

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: D

Enter Grade: D

Enter Grade: D

Enter Grade: D

Enter Grade: A

Enter Grade: A

Enter Grade: B

**Enter Student ID:** 27

**Enter Student Name:** STUDENT C2

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: B

**Enter Student ID:** 6

**Enter Student Name:** STUDENT F

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

**Enter Student ID:** 2

**Enter Student Name:** STUDENT B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

**Enter Student ID:** 15

**Enter Student Name:** STUDENT O

Enter Grade: B

Enter Grade: C

Enter Grade: B

Enter Grade: C

Enter Grade: B

Enter Grade: C

Enter Grade: B

Enter Grade: C

Enter Grade: A

Enter Grade: A

**Enter Student ID:** 7

**Enter Student Name:** STUDENT G

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

**Enter Student ID:** 24

**Enter Student Name:** STUDENT X

Enter Grade: C

Enter Grade: A

Enter Grade: C

Enter Grade: A

Enter Grade: C

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: B

**Enter Student ID:** 9

**Enter Student Name:** STUDENT I

Enter Grade: C

Enter Grade: B

Enter Grade: A

Enter Grade: B

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: A

Enter Grade: B

Enter Grade: B

**Enter Student ID:** 22

**Enter Student Name:** STUDENT V

Enter Grade: C

Enter Grade: C

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: D

Enter Grade: D

Enter Grade: B

Enter Grade: A

**Enter Student ID:** 5

**Enter Student Name:** STUDENT E

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

**Enter Student ID:** 23

**Enter Student Name:** STUDENT W

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: A

**Enter Student ID:** 12

**Enter Student Name:** STUDENT L

Enter Grade: C

Enter Grade: A

Enter Grade: B

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: D

Enter Grade: A

Enter Grade: B

Enter Grade: C

**Enter Student ID:** 8

**Enter Student Name:** STUDENT H

Enter Grade: C

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: B

Enter Grade: A

Enter Grade: C

Enter Grade: D

**Enter Student ID:** 10

**Enter Student Name:** STUDENT J

Enter Grade: A

Enter Grade: B

Enter Grade: C

Enter Grade: D

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: D

Enter Grade: C

Enter Grade: A

**Enter Student ID:** 18

**Enter Student Name:** STUDENT R

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: C

**Enter Student ID:** 0

**Enter Student Name:** STUDENT M9

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: C

**Enter Student ID:** 20

**Enter Student Name:** STUDENT T

Enter Grade: A

Enter Grade: B

Enter Grade: C

Enter Grade: A

Enter Grade: B

Enter Grade: C

Enter Grade: A

Enter Grade: B

Enter Grade: C

Enter Grade: S

**Enter Student ID:** 21

**Enter Student Name:** STUDENT U

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

**Enter Student ID:** 19

**Enter Student Name:** STUDENT S

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: A

Enter Grade: D

Enter Grade: C

**Enter Student ID:** 1

**Enter Student Name:** STUDENT A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: B

Enter Grade: C

Enter Grade: A

Enter Grade: B

**Enter Student ID:** 11

**Enter Student Name:** STUDENT K

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: C

Enter Grade: C

Enter Grade: B

Enter Grade: B

Enter Grade: B

**1) Unsorted List of Student Records:**

[[16, 'STUDENT P', 'p'], [25, 'STUDENT Y', 'p'], [26, 'STUDENT Z1', 'p'], [3, 'STUDENT C', 'p'], [14, 'STUDENT N', 'p'], [13, 'STUDENT M', 'p'], [4, 'STUDENT D', 'p'], [29, 'STUDENT Y6', 'p'], [28, 'STUDENT P4', 'p'], [17, 'STUDENT Q', 'p'], [27, 'STUDENT C2', 'p'], [6, 'STUDENT F', 'p'], [2, 'STUDENT B', 'p'], [15, 'STUDENT O', 'p'], [7, 'STUDENT G', 'p'], [24, 'STUDENT X', 'p'], [9, 'STUDENT I', 'p'], [22, 'STUDENT V', 'p'], [5, 'STUDENT E', 'p'], [23, 'STUDENT W', 'p'], [12, 'STUDENT L', 'p'], [8, 'STUDENT H', 'p'], [10, 'STUDENT J', 'p'], [18, 'STUDENT R', 'p'], [0, 'STUDENT M9', 'p'], [20, 'STUDENT T', 'p'], [21, 'STUDENT U', 'p'], [19, 'STUDENT S', 'p'], [1, 'STUDENT A', 'p'], [11, 'STUDENT K', 'p']]

**2) In order Traversal of the Binary Search Tree:**

Student ID: 0

Student Name: STUDENT M9

Grades: ['A', 'A', 'A', 'A', 'A', 'C', 'C', 'B', 'B', 'B']

Student ID: 1

Student Name: STUDENT A

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'C', 'C', 'A', 'B']

Student ID: 2

Student Name: STUDENT B

Grades: ['B', 'B', 'A', 'A', 'A', 'A', 'C', 'C', 'D', 'A']

Student ID: 3

Student Name: STUDENT C

Grades: ['A', 'A', 'A', 'C', 'C', 'C', 'V', 'B', 'B', 'A']

Student ID: 4

Student Name: STUDENT D

Grades: ['A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A']

Student ID: 5

Student Name: STUDENT E

Grades: ['B', 'B', 'D', 'A', 'A', 'D', 'C', 'A', 'B', 'A']

Student ID: 6

Student Name: STUDENT F

Grades: ['C', 'C', 'D', 'D', 'D', 'A', 'A', 'B', 'B', 'C']

Student ID: 7

Student Name: STUDENT G

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'C']

Student ID: 8

Student Name: STUDENT H

Grades: ['A', 'B', 'A', 'B', 'A', 'B', 'A', 'B', 'A', 'A']

Student ID: 9

Student Name: STUDENT I

Grades: ['B', 'B', 'A', 'A', 'A', 'C', 'C', 'C', 'A', 'C']

Student ID: 10

Student Name: STUDENT J

Grades: ['C', 'C', 'C', 'D', 'D', 'D', 'D', 'A', 'A', 'B']

Student ID: 11

Student Name: STUDENT K

Grades: ['A', 'A', 'A', 'A', 'C', 'C', 'C', 'C', 'C', 'B']

Student ID: 12

Student Name: STUDENT L

Grades: ['A', 'A', 'A', 'C', 'C', 'C', 'B', 'B', 'B', 'B']

Student ID: 13

Student Name: STUDENT M

Grades: ['B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'C', 'C']

Student ID: 14

Student Name: STUDENT N

Grades: ['B', 'C', 'B', 'C', 'B', 'C', 'B', 'C', 'A', 'A']

Student ID: 15

Student Name: STUDENT O

Grades: ['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']

Student ID: 16

Student Name: STUDENT P

Grades: ['C', 'A', 'C', 'A', 'C', 'A', 'B', 'B', 'A', 'B']

Student ID: 17

Student Name: STUDENT Q

Grades: ['C', 'B', 'A', 'B', 'A', 'C', 'C', 'A', 'B', 'B']

Student ID: 18

Student Name: STUDENT R

Grades: ['C', 'C', 'A', 'A', 'A', 'A', 'D', 'D', 'B', 'A']

Student ID: 19

Student Name: STUDENT S

Grades: ['C', 'C', 'C', 'A', 'A', 'A', 'A', 'B', 'B', 'B']

Student ID: 20

Student Name: STUDENT T

Grades: ['A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'C', 'A']

Student ID: 21

Student Name: STUDENT U

Grades: ['C', 'A', 'B', 'A', 'C', 'C', 'D', 'A', 'B', 'C']

Student ID: 22

Student Name: STUDENT V

Grades: ['C', 'B', 'B', 'A', 'A', 'C', 'B', 'A', 'C', 'D']

Student ID: 23

Student Name: STUDENT W

Grades: ['A', 'B', 'C', 'D', 'A', 'B', 'B', 'D', 'C', 'A']

Student ID: 24

Student Name: STUDENT X

Grades: ['C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C']

Student ID: 25

Student Name: STUDENT Y

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'C']

Student ID: 26

Student Name: STUDENT Z1

Grades: ['A', 'B', 'C', 'A', 'B', 'C', 'A', 'B', 'C', 'S']

Student ID: 27

Student Name: STUDENT C2

Grades: ['B', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'C']

Student ID: 28

Student Name: STUDENT P4

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'C', 'A', 'D', 'C']

Student ID: 29

Student Name: STUDENT Y6

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'A', 'B']

**Enter First ID to be searched:** 11

**Enter Second ID to be searched:** 28

**3) First ID Found:**

Student ID: 11

Student Name: STUDENT K

Grades: ['A', 'A', 'A', 'A', 'C', 'C', 'C', 'C', 'C', 'B']

**3) Second ID Found:**

Student ID: 28

Student Name: STUDENT P4

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'C', 'A', 'D', 'C']

**Enter ID to be deleted:** 9

**Enter ID to be deleted:** 6

**4) In order Traversal of the Binary Search Tree After Deleting two nodes:**

Student ID: 0

Student Name: STUDENT M9

Grades: ['A', 'A', 'A', 'A', 'A', 'C', 'C', 'B', 'B', 'B']

Student ID: 1

Student Name: STUDENT A

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'C', 'C', 'A', 'B']

Student ID: 2

Student Name: STUDENT B

Grades: ['B', 'B', 'A', 'A', 'A', 'A', 'C', 'C', 'D', 'A']

Student ID: 3

Student Name: STUDENT C

Grades: ['A', 'A', 'A', 'C', 'C', 'C', 'V', 'B', 'B', 'A']

Student ID: 4

Student Name: STUDENT D

Grades: ['A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A']

Student ID: 5

Student Name: STUDENT E

Grades: ['B', 'B', 'D', 'A', 'A', 'D', 'C', 'A', 'B', 'A']

Student ID: 7

Student Name: STUDENT G

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'C']

Student ID: 8

Student Name: STUDENT H

Grades: ['A', 'B', 'A', 'B', 'A', 'B', 'A', 'B', 'A', 'A']

Student ID: 10

Student Name: STUDENT J

Grades: ['C', 'C', 'C', 'D', 'D', 'D', 'D', 'A', 'A', 'B']

Student ID: 11

Student Name: STUDENT K

Grades: ['A', 'A', 'A', 'A', 'C', 'C', 'C', 'C', 'C', 'B']

Student ID: 12

Student Name: STUDENT L

Grades: ['A', 'A', 'A', 'C', 'C', 'C', 'B', 'B', 'B', 'B']

Student ID: 13

Student Name: STUDENT M

Grades: ['B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'C', 'C']

Student ID: 14

Student Name: STUDENT N

Grades: ['B', 'C', 'B', 'C', 'B', 'C', 'B', 'C', 'A', 'A']

Student ID: 15

Student Name: STUDENT O

Grades: ['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']

Student ID: 16

Student Name: STUDENT P

Grades: ['C', 'A', 'C', 'A', 'C', 'A', 'B', 'B', 'A', 'B']

Student ID: 17

Student Name: STUDENT Q

Grades: ['C', 'B', 'A', 'B', 'A', 'C', 'C', 'A', 'B', 'B']

Student ID: 18

Student Name: STUDENT R

Grades: ['C', 'C', 'A', 'A', 'A', 'A', 'D', 'D', 'B', 'A']

Student ID: 19

Student Name: STUDENT S

Grades: ['C', 'C', 'C', 'A', 'A', 'A', 'A', 'B', 'B', 'B']

Student ID: 20

Student Name: STUDENT T

Grades: ['A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'C', 'A']

Student ID: 21

Student Name: STUDENT U

Grades: ['C', 'A', 'B', 'A', 'C', 'C', 'D', 'A', 'B', 'C']

Student ID: 22

Student Name: STUDENT V

Grades: ['C', 'B', 'B', 'A', 'A', 'C', 'B', 'A', 'C', 'D']

Student ID: 23

Student Name: STUDENT W

Grades: ['A', 'B', 'C', 'D', 'A', 'B', 'B', 'D', 'C', 'A']

Student ID: 24

Student Name: STUDENT X

Grades: ['C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C']

Student ID: 25

Student Name: STUDENT Y

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'C']

Student ID: 26

Student Name: STUDENT Z1

Grades: ['A', 'B', 'C', 'A', 'B', 'C', 'A', 'B', 'C', 'S']

Student ID: 27

Student Name: STUDENT C2

Grades: ['B', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'C']

Student ID: 28

Student Name: STUDENT P4

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'C', 'A', 'D', 'C']

Student ID: 29

Student Name: STUDENT Y6

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'A', 'B']

**Enter Student ID:** 40

**Enter Student Name:** STUDENT XYZ

Enter Grade: B

Enter Grade: A

Enter Grade: A

Enter Grade: A

Enter Grade: B

Enter Grade: C

Enter Grade: C

Enter Grade: C

Enter Grade: A

Enter Grade: B

**5) In order Traversal of the Binary Search Tree after Inserting one node:**

Student ID: 0

Student Name: STUDENT M9

Grades: ['A', 'A', 'A', 'A', 'A', 'C', 'C', 'B', 'B', 'B']

Student ID: 1

Student Name: STUDENT A

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'C', 'C', 'A', 'B']

Student ID: 2

Student Name: STUDENT B

Grades: ['B', 'B', 'A', 'A', 'A', 'A', 'C', 'C', 'D', 'A']

Student ID: 3

Student Name: STUDENT C

Grades: ['A', 'A', 'A', 'C', 'C', 'C', 'V', 'B', 'B', 'A']

Student ID: 4

Student Name: STUDENT D

Grades: ['A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A']

Student ID: 5

Student Name: STUDENT E

Grades: ['B', 'B', 'D', 'A', 'A', 'D', 'C', 'A', 'B', 'A']

Student ID: 7

Student Name: STUDENT G

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'C']

Student ID: 8

Student Name: STUDENT H

Grades: ['A', 'B', 'A', 'B', 'A', 'B', 'A', 'B', 'A', 'A']

Student ID: 10

Student Name: STUDENT J

Grades: ['C', 'C', 'C', 'D', 'D', 'D', 'D', 'A', 'A', 'B']

Student ID: 11

Student Name: STUDENT K

Grades: ['A', 'A', 'A', 'A', 'C', 'C', 'C', 'C', 'C', 'B']

Student ID: 12

Student Name: STUDENT L

Grades: ['A', 'A', 'A', 'C', 'C', 'C', 'B', 'B', 'B', 'B']

Student ID: 13

Student Name: STUDENT M

Grades: ['B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'C', 'C']

Student ID: 14

Student Name: STUDENT N

Grades: ['B', 'C', 'B', 'C', 'B', 'C', 'B', 'C', 'A', 'A']

Student ID: 15

Student Name: STUDENT O

Grades: ['B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']

Student ID: 16

Student Name: STUDENT P

Grades: ['C', 'A', 'C', 'A', 'C', 'A', 'B', 'B', 'A', 'B']

Student ID: 17

Student Name: STUDENT Q

Grades: ['C', 'B', 'A', 'B', 'A', 'C', 'C', 'A', 'B', 'B']

Student ID: 18

Student Name: STUDENT R

Grades: ['C', 'C', 'A', 'A', 'A', 'A', 'D', 'D', 'B', 'A']

Student ID: 19

Student Name: STUDENT S

Grades: ['C', 'C', 'C', 'A', 'A', 'A', 'A', 'B', 'B', 'B']

Student ID: 20

Student Name: STUDENT T

Grades: ['A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'C', 'A']

Student ID: 21

Student Name: STUDENT U

Grades: ['C', 'A', 'B', 'A', 'C', 'C', 'D', 'A', 'B', 'C']

Student ID: 22

Student Name: STUDENT V

Grades: ['C', 'B', 'B', 'A', 'A', 'C', 'B', 'A', 'C', 'D']

Student ID: 23

Student Name: STUDENT W

Grades: ['A', 'B', 'C', 'D', 'A', 'B', 'B', 'D', 'C', 'A']

Student ID: 24

Student Name: STUDENT X

Grades: ['C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C']

Student ID: 25

Student Name: STUDENT Y

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'C']

Student ID: 26

Student Name: STUDENT Z1

Grades: ['A', 'B', 'C', 'A', 'B', 'C', 'A', 'B', 'C', 'S']

Student ID: 27

Student Name: STUDENT C2

Grades: ['B', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'A', 'C']

Student ID: 28

Student Name: STUDENT P4

Grades: ['A', 'A', 'A', 'A', 'B', 'B', 'C', 'A', 'D', 'C']

Student ID: 29

Student Name: STUDENT Y6

Grades: ['A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'A', 'B']

Student ID: 40

Student Name: STUDENT XYZ

Grades: ['B', 'A', 'A', 'A', 'B', 'C', 'C', 'C', 'A', 'B']

**9.3 SOURCE PROGRAM**

Implemented in **Python** programming language.

import os

x = [[] for i in range(25)]

l = []

s = []

class Node:

def \_\_init\_\_(self, key, name, grades):

self.name = name

self.key = key

self.grades = grades

self.leftChild = None

self.rightChild = None

def printData(self):

print("\n Student ID: " + str(self.key) + "\n Student Name: " + self.name + "\n Grades: " + str(x[self.key -1]))

class BinarySearchTree:

def \_\_init\_\_(self):

self.root = None

def addNode(self, key, name, grades):

newNode = Node(key, name, grades)

if self.root == None:

self.root = newNode

else:

focusNode = self.root

while(True):

parent = focusNode

# left branch

if key < focusNode.key:

focusNode = focusNode.leftChild

if focusNode == None:

parent.leftChild = newNode

return

# right branch

else:

focusNode = focusNode.rightChild

if focusNode == None:

parent.rightChild = newNode

return

def inOrderTraverseTree(self, focusNode):

if focusNode != None:

self.inOrderTraverseTree(focusNode.leftChild)

focusNode.printData()

self.inOrderTraverseTree(focusNode.rightChild)

def findNode(self, key):

focusNode = self.root

while focusNode.key != key:

if key < focusNode.key:

focusNode = focusNode.leftChild

else:

focusNode = focusNode.rightChild

if focusNode == None:

return None

focusNode.printData()

def getReplacementNode(self, replacedNode):

replacementParent = replacedNode

replacement = replacedNode

focusNode = replacedNode.rightChild

# move nodes

while(focusNode != None):

replacementParent = replacement

replacement = focusNode

focusNode = focusNode.leftChild

if replacement != replacedNode.rightChild:

replacementParent.leftChild = replacement.rightChild

replacement.rightChild = replacedNode.rightChild

return replacement

def removeNode(self, key):

focusNode = self.root

parent = self.root

isLeftChild = True

isRightChild = True

while(focusNode.key != key):

parent = focusNode

# go left

if(key < focusNode.key):

isLeftChild = True

focusNode = focusNode.leftChild

# go right

else:

isLeftChild = False

focusNode = focusNode.rightChild

if(focusNode == None):

return False

if focusNode.leftChild == None and focusNode.rightChild == None:

if focusNode == self.root:

self.root = None

elif isLeftChild:

parent.leftChild = None

else:

parent.rightChild = None

# no right child

elif focusNode.rightChild == None:

if focusNode == self.root:

root = focusNode.leftChild

elif isLeftChild:

parent.leftChild = focusNode.leftChild

else:

parent.rightChild = focusNode.leftChild

# no left child

elif focusNode.leftChild == None:

if focusNode == self.root:

root = focusNode.rightChild

elif isRightchild:

parent.leftChild = focusNode.rightCHild

else:

parent.rightChild = focusNode.rightChild

else:

getReplacementNode(focusNode)

if focusNode == self.root:

root = replacement

elif isLeftChild:

parent.leftChild = replacement

else:

parent.rightChild = replacement

replacement.leftchild = focusNode.leftChild

return True

if \_\_name\_\_ == "\_\_main\_\_":

b = BinarySearchTree()

n = int(input("\n Enter total number of students: "))

for i in range(n):

s\_id = int(input("\n Enter Student ID: "))

s\_name = input("\n Enter Student Name: ")

for j in range(10):

g = input("\n Enter Grade: ")

l.append(g)

x[i] = l

u = [s\_id,s\_name,"p"]

s.append(u)

l = []

print("\n 1) Unsorted List of Student Records: ")

print("\n ",s)

for k in range(len(s)):

b.addNode(s[k][0], s[k][1], s[k][2])

print("\n 2) In order Traversal of the Binary Search Tree: ")

b.inOrderTraverseTree(b.root)

f1 = int(input("\n Enter First ID to be searched: "))

f2 = int(input("\n Enter Second ID to be searched: "))

print("\n 3) First ID Found: ")

b.findNode(f1)

print("\n 3) Second ID Found: ")

b.findNode(f2)

d1 = int(input("\n Enter ID to be deleted: "))

d2 = int(input("\n Enter ID to be deleted: "))

b.removeNode(d1)

b.removeNode(d2)

print(" \n 4) In order Traversal of the Binary Search Tree After Deleting two nodes: ")

b.inOrderTraverseTree(b.root)

s\_id1 = int(input("\n Enter Student ID: "))

s\_name1 = input("\n Enter Student Name: ")

for j in range(10):

g = input("\n Enter Grade: ")

l.append(g)

x[i] = l

b.addNode(s\_id1, s\_name1, "p")

l = []

print("\n 5) In order Traversal of the Binary Search Tree after Inserting one node: ")

b.inOrderTraverseTree(b.root)

**CHAPTER 10**

**GRAPH ALGORITHM AND BREADTH FIRST SEARCH TREE**

**10.1 PROBLEM DESCRIPTION AND EXPLANATION**

Breadth-first search is one of the simplest algorithms for searching a graph and the archetype for many important graph algorithms. Prim’s minimum-spanningtree algorithm and Dijkstra’s single-source shortest-paths algorithm use ideas similar to those in breadth-first search. Given a graph G = (V,E) and a distinguished source vertex s, breadth-first search systematically explores the edges of G to “discover” every vertex that is reachable from s. It computes the distance (smallest number of edges) from s to each reachable vertex. It also produces a “breadth-first tree” with root s that contains all reachable vertices. For any vertex v reachable from s, the simple path in the breadth-first tree from s to v corresponds to a “shortest path” from s to v in G, that is, a path containing the smallest number of edges. The algorithm works on both directed and undirected graphs.

Breadth-first search constructs a breadth-first tree, initially containing only its root, which is the source vertex s. Whenever the search discovers a white vertex v in the course of scanning the adjacency list of an already discovered vertex u, the vertex v and the edge (u, v) are added to the tree. We say that u is the predecessor or parent of v in the breadth-first tree. Since a vertex is discovered at most once, it has at most one parent. Ancestor and descendant relationships in the breadth-first tree are defined relative to the root s as usual. If u is on the simple path in the tree from the root s to vertex v, then u is an ancestor of v and v is a descendant of u.

The breadth-first-search procedure BFS below assumes that the input graph G = (V,E) is represented using adjacency lists. It attaches several additional attributes to each vertex in the graph. We store the color of each vertex u ϵ V in the attribute u.color and the predecessor of u in the attribute u.π. If u has no predecessor (for example, if u = s or u has not been discovered), then u.π = NIL. The attribute u.d holds the distance from the source s to vertex u computed by the algorithm. The algorithm also uses a first-in, first-out queue Q to manage the set of gray vertices.

BFS(G, s)

1 for each vertex u ϵ G.V - { s }

2 u.color = WHITE

3 u.d = ∞

4 u.π = NIL

5 s.color = GRAY

6 s.d = 0

7 s.π = NIL

8 Q = ø ;

9 ENQUEUE(Q, s)

10 while Q ≠ ø;

11 u = DEQUEUE(Q)

12 for each v ϵ G.Adj[u]

13 if v.color == WHITE

14 v.color = GRAY

15 v.d = u:d + 1

16 v.π = u

17 ENQUEUE(Q, v)

18 u.color = BLACK

**10.2 TABULATION, OUTPUT AND ANALYSIS**

Python 3.1 (r31:73574, Jun 26 2009, 17:50:52) [MSC v.1500 64 bit (AMD64)] on win32

Type "copyright", "credits" or "license()" for more information.

>>> ================================ RESTART ================================

>>>

**Enter total number of vertices:** 10

Enter the Vertex: A

Enter total number of edges: 3

Enter edge 1: A

Enter edge 2: B

Enter edge 1: A

Enter edge 2: D

Enter edge 1: A

Enter edge 2: E

Enter the Vertex: B

Enter total number of edges: 2

Enter edge 1: B

Enter edge 2: C

Enter edge 1: B

Enter edge 2: E

Enter the Vertex: C

Enter total number of edges: 3

Enter edge 1: C

Enter edge 2: B

Enter edge 1: C

Enter edge 2: F

Enter edge 1: C

Enter edge 2: J

Enter the Vertex: D

Enter total number of edges: 2

Enter edge 1: D

Enter edge 2: A

Enter edge 1: D

Enter edge 2: H

Enter the Vertex: E

Enter total number of edges: 2

Enter edge 1: E

Enter edge 2: A

Enter edge 1: E

Enter edge 2: D

Enter the Vertex: F

Enter total number of edges: 2

Enter edge 1: F

Enter edge 2: B

Enter edge 1: F

Enter edge 2: I

Enter the Vertex: G

Enter total number of edges: 1

Enter edge 1: G

Enter edge 2: C

Enter the Vertex: H

Enter total number of edges: 2

Enter edge 1: H

Enter edge 2: E

Enter edge 1: H

Enter edge 2: I

Enter the Vertex: I

Enter total number of edges: 1

Enter edge 1: I

Enter edge 2: E

Enter the Vertex: J

Enter total number of edges: 3

Enter edge 1: J

Enter edge 2: C

Enter edge 1: J

Enter edge 2: G

Enter edge 1: J

Enter edge 2: I

**10.2.1 GRAPH Created --- After inserting 10 elements:**

**{ 'A' : ['B', 'D', 'E'],**

**'C': ['B', 'F', 'J'],**

**'B': ['C', 'E'],**

**'E': ['A', 'D'],**

**'D': ['A', 'H'],**

**'G': ['C'],**

**'F' : ['B', 'I'],**

**'I' : ['E'],**

**'H': ['E', 'I'],**

**'J': ['C', 'G', 'I'] }**

**Vertices of graph:**

['A', 'C', 'B', 'E', 'D', 'G', 'F', 'I', 'H', 'J']

**Edges of graph:**

[{'A', 'B'}, {'A', 'D'}, {'A', 'E'}, {'C', 'B'}, {'C', 'F'}, {'C', 'J'}, {'B', 'E'}, {'E', 'D'}, {'H', 'D'}, {'C', 'G'}, {'B', 'F'}, {'I', 'F'}, {'I', 'E'}, {'H', 'E'}, {'I', 'H'}, {'J', 'G'}, {'I', 'J'}]

**Enter vertex to be searched:** G

**Enter vertex to be searched:** J

**Enter vertex to be searched:** I

**10.2.2 Root Node A -- Vertex G**

**10.2.2.1 Complete Tree:**

['A']

['A', 'B']

['A', 'D']

['A', 'E']

['A', 'B', 'C']

['A', 'B', 'E']

['A', 'D', 'H']

['A', 'E', 'D']

['A', 'B', 'C', 'F']

['A', 'B', 'C', 'J']

['A', 'B', 'E', 'D']

['A', 'D', 'H', 'E']

['A', 'D', 'H', 'I']

['A', 'E', 'D', 'H']

['A', 'B', 'C', 'F', 'I']

**['A', 'B', 'C', 'J', 'G']**

['A', 'B', 'C', 'J', 'I']

['A', 'B', 'E', 'D', 'H']

['A', 'D', 'H', 'I', 'E']

['A', 'E', 'D', 'H', 'I']

['A', 'B', 'C', 'F', 'I', 'E']

['A', 'B', 'C', 'J', 'I', 'E']

['A', 'B', 'E', 'D', 'H', 'I']

['A', 'B', 'C', 'F', 'I', 'E', 'D']

['A', 'B', 'C', 'J', 'I', 'E', 'D']

['A', 'B', 'C', 'F', 'I', 'E', 'D', 'H']

['A', 'B', 'C', 'J', 'I', 'E', 'D', 'H']

**10.2.2.2 SHORTEST PATH of the element from the Root Node :**

['A', 'B', 'C', 'J', 'G']

**10.2.3 Root Node A -- Vertex J**

**10.2.3.1 Complete Tree:**

['A']

['A', 'B']

['A', 'D']

['A', 'E']

['A', 'B', 'C']

['A', 'B', 'E']

['A', 'D', 'H']

['A', 'E', 'D']

['A', 'B', 'C', 'F']

**['A', 'B', 'C', 'J']**

['A', 'B', 'E', 'D']

['A', 'D', 'H', 'E']

['A', 'D', 'H', 'I']

['A', 'E', 'D', 'H']

['A', 'B', 'C', 'F', 'I']

['A', 'B', 'C', 'J', 'G']

['A', 'B', 'C', 'J', 'I']

['A', 'B', 'E', 'D', 'H']

['A', 'D', 'H', 'I', 'E']

['A', 'E', 'D', 'H', 'I']

['A', 'B', 'C', 'F', 'I', 'E']

['A', 'B', 'C', 'J', 'I', 'E']

['A', 'B', 'E', 'D', 'H', 'I']

['A', 'B', 'C', 'F', 'I', 'E', 'D']

['A', 'B', 'C', 'J', 'I', 'E', 'D']

['A', 'B', 'C', 'F', 'I', 'E', 'D', 'H']

['A', 'B', 'C', 'J', 'I', 'E', 'D', 'H']

**10.2.3.2 SHORTEST PATH of the element from the Root Node :**

['A', 'B', 'C', 'J']

**10.2.4 Root Node A -- Vertex I**

**10.2.4.1 Complete Tree:**

['A']

['A', 'B']

['A', 'D']

['A', 'E']

['A', 'B', 'C']

['A', 'B', 'E']

['A', 'D', 'H']

['A', 'E', 'D']

['A', 'B', 'C', 'F']

['A', 'B', 'C', 'J']

['A', 'B', 'E', 'D']

['A', 'D', 'H', 'E']

**['A', 'D', 'H', 'I']**

['A', 'E', 'D', 'H']

['A', 'B', 'C', 'F', 'I']

['A', 'B', 'C', 'J', 'G']

['A', 'B', 'C', 'J', 'I']

['A', 'B', 'E', 'D', 'H']

['A', 'D', 'H', 'I', 'E']

['A', 'E', 'D', 'H', 'I']

['A', 'B', 'C', 'F', 'I', 'E']

['A', 'B', 'C', 'J', 'I', 'E']

['A', 'B', 'E', 'D', 'H', 'I']

['A', 'B', 'C', 'F', 'I', 'E', 'D']

['A', 'B', 'C', 'J', 'I', 'E', 'D']

['A', 'B', 'C', 'F', 'I', 'E', 'D', 'H']

['A', 'B', 'C', 'J', 'I', 'E', 'D', 'H']

**10.2.4.2 SHORTEST PATH of the element from the Root Node :**

['A', 'D', 'H', 'I']

>>>

**10.3 SOURCE PROGRAM**

Implemented in **Python** programming language.

import os

class Graph(object):

def \_\_init\_\_(self, graph\_dict={}):

""" initializes a graph object """

self.\_\_graph\_dict = graph\_dict

def vertices(self):

""" returns the vertices of a graph """

return list(self.\_\_graph\_dict.keys())

def edges(self):

""" returns the edges of a graph """

return self.\_\_generate\_edges()

def add\_vertex(self, vertex):

""" If the vertex "vertex" is not in

self.\_\_graph\_dict, a key "vertex" with an empty

list as a value is added to the dictionary.

Otherwise nothing has to be done.

"""

if vertex not in self.\_\_graph\_dict:

self.\_\_graph\_dict[vertex] = []

def add\_edge(self, edge):

""" assumes that edge is of type set, tuple or list;

between two vertices can be multiple edges!

"""

self.\_\_graph\_dict[edge[0]].append(edge[1])

def \_\_generate\_edges(self):

""" A static method generating the edges of the

graph "graph". Edges are represented as sets

with one (a loop back to the vertex) or two

vertices

"""

edges = []

for vertex in self.\_\_graph\_dict:

for neighbour in self.\_\_graph\_dict[vertex]:

if {neighbour, vertex} not in edges:

edges.append({vertex, neighbour})

return edges

def \_\_str\_\_(self):

res = "vertices: "

for k in self.\_\_graph\_dict:

res += str(k) + " "

res += "\nedges: "

for edge in self.\_\_generate\_edges():

res += str(edge) + " "

return res

class MyQUEUE: # just an implementation of a queue

def \_\_init\_\_(self):

self.holder = []

def enqueue(self,val):

self.holder.append(val)

def dequeue(self):

val = None

try:

val = self.holder[0]

if len(self.holder) == 1:

self.holder = []

else:

self.holder = self.holder[1:]

except:

pass

return val

def IsEmpty(self):

result = False

if len(self.holder) == 0:

result = True

return result

if \_\_name\_\_ == "\_\_main\_\_":

# Initializing Graph

graph = {}

# Creating a Graph Object

gra = Graph(graph)

n = int(input("\n Enter total number of vertices: "))

for i in range(n):

v = input("\n Enter the Vertex: ")

gra.add\_vertex(v)

m = int(input("\n Enter total number of edges: "))

for j in range(m):

e1 = input("\n Enter edge 1: ")

e2 = input("\n Enter edge 2: ")

gra.add\_edge([e1,e2])

print("\n GRAPH Created --- After inserting 10 elements: ")

print("\n",graph)

print("\n Vertices of graph:")

print("\n",gra.vertices())

print("\n Edges of graph:")

print("\n",gra.edges())

path\_queue = MyQUEUE() # now we make a queue

def BFS(graph,start,end,q):

temp\_path = [start]

q.enqueue(temp\_path)

print("\n\n Complete Tree: ")

while q.IsEmpty() == False:

tmp\_path = q.dequeue()

last\_node = tmp\_path[len(tmp\_path)-1]

print (tmp\_path)

if last\_node == end:

print ("\n SHORTEST PATH of the element from the Root Node : ")

print("\n",tmp\_path)

for link\_node in graph[last\_node]:

if link\_node not in tmp\_path:

new\_path = []

new\_path = tmp\_path + [link\_node]

q.enqueue(new\_path)

v1 = input("\n Enter vertex to be searched: ")

v2 = input("\n Enter vertex to be searched: ")

v3 = input("\n Enter vertex to be searched: ")

print("\n\n 1) Root Node A -- Vertex ",v1)

BFS(graph,"A",v1,path\_queue)

print("\n\n 2) Root Node A -- Vertex ",v2)

BFS(graph,"A",v2,path\_queue)

print("\n\n 3) Root Node A -- Vertex ",v3)

BFS(graph,"A",v3,path\_queue)

**CHAPTER 11**

**CONCLUSION**

In **Insertion Sort algorithm**, we sort the elements of an array by comparing consecutive elements.

In **Merge Sort algorithm,** we divide the array of elements into two arrays and each divided array into two more arrays and we keep dividing until there is only one element in each array, then we sort the elements and merge the arrays till we get sorted array of elements.

**Quick Sort algorithm** is used commonly when there is a large input of elements.

**Heap Sort algorithm,** creates a heap of elements where both the child nodes are greater than the parent node and then sorts the elements.

**Hashing** technique is used to store the values of the elements in a data structure without creating collisions.

The **Huffman coding** **algorithm** makes use of the Greedy property in order to compress the text files by assigning binary code to each unique character of the text file by the order of its frequency.

**Priority Queue** makes use of the Heapify function of Heap Sort to schedule jobs based on their priority for efficient performance and minimum consumption of resources.

**Dynamic Programming** is used to find an optimal way to multiply a set of matrices by comparing their scalar multiplications.

**Binary Search Tree** is used to store records as a tree where the left child is less than the parent and the right child is greater than the parent. In-order traversal of the tree yields the sorted order of records.

**Graph Algorithm** is used to find shortest path between two nodes in a network to transmit data in less time.