**DAA-Lab**

**LAB - 6**

LQ1. Aim of the experiment : To implement Heap Sort. And count total number of comparisons. Input : 792 11 19 17 12 5 7 12 Expected output : Copy : [792 11 19 17 12 5 7 12 ] Heap : [ 19 12 17 11 9 2 12 5 77] Heapsort : [257 79 11 12 12 17 19 ] 9 comparisons

CASE-1

**INPUT:**

Array: 30 22 15 10 27 18 35 12

**OUTPUT:**

Copy: [30, 22, 15, 10, 27, 18, 35, 12]

Heap: [10, 12, 15, 22, 27, 18, 35, 30]

Heapsort: [10, 12, 15, 18, 22, 27, 30, 35]

Total comparisons: 20

CASE-2

**INPUT**

Array: 8 4 16 2 10 6 12 14

**OUTPUT:**

Copy: [8, 4, 16, 2, 10, 6, 12, 14]

Heap: [2, 4, 6, 8, 10, 16, 12, 14]

Heapsort: [2, 4, 6, 8, 10, 12, 14, 16]

Total comparisons: 18

CASE-3

**INPUT:**

Array: 5 8 15 2 10 7 13 12

**OUTPUT:**

Copy: [5, 8, 15, 2, 10, 7, 13, 12]

Heap: [2, 5, 7, 8, 10, 15, 13, 12]

Heapsort: [2, 5, 7, 8, 10, 12, 13, 15]

Total comparisons: 16

LQ2. Aim of the experiment : To implement a menu (given as follows) driven program to sort an array of n integers in ascending order by heap sort algorithm and perform the operations on max heap. Determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the array to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

MAX-HEAP & PRIORITY QUEUE MENU

0. Quit

1. n Random numbers => Array

2. Display the Array

3. Sort the Array in Ascending Order by using Max-Heap Sort technique

4. Sort the Array in Descending Order by using any algorithm

5. Time Complexity to sort ascending of random data

6. Time Complexity to sort ascending of data already sorted in ascending order

7. Time Complexity to sort ascending of data already sorted in descending order

8. Time Complexity to sort ascending all Cases (Data Ascending, Data in Descending & Random Data) in Tabular form for values n=5000 to 50000, step=5000

9. Extract largest element

10. Replace value at a node with a new value

11. Insert a new element

12. Delete an element

Enter your choice: 1

Enter the number of random numbers: 10

MAX-HEAP & PRIORITY QUEUE MENU

0. Quit

1. n Random numbers => Array

2. Display the Array

3. Sort the Array in Ascending Order by using Max-Heap Sort technique

4. Sort the Array in Descending Order by using any algorithm

5. Time Complexity to sort ascending of random data

6. Time Complexity to sort ascending of data already sorted in ascending order

7. Time Complexity to sort ascending of data already sorted in descending order

8. Time Complexity to sort ascending all Cases (Data Ascending, Data in Descending & Random Data) in Tabular form for values n=5000 to 50000, step=5000

9. Extract largest element

10. Replace value at a node with a new value

11. Insert a new element

12. Delete an element

Enter your choice: 2

Array: [43, 78, 24, 92, 67, 53, 14, 85, 91, 1]

MAX-HEAP & PRIORITY QUEUE MENU

0. Quit

1. n Random numbers => Array

2. Display the Array

3. Sort the Array in Ascending Order by using Max-Heap Sort technique

4. Sort the Array in Descending Order by using any algorithm

5. Time Complexity to sort ascending of random data

6. Time Complexity to sort ascending of data already sorted in ascending order

7. Time Complexity to sort ascending of data already sorted in descending order

8. Time Complexity to sort ascending all Cases (Data Ascending, Data in Descending & Random Data) in Tabular form for values n=5000 to 50000, step=5000

9. Extract largest element

10. Replace value at a node with a new value

11. Insert a new element

12. Delete an element

Enter your choice: 3

Array sorted in ascending order using Max-Heap Sort: [1, 14, 24, 43, 53, 67, 78, 85, 91, 92]

MAX-HEAP & PRIORITY QUEUE MENU

0. Quit

1. n Random numbers => Array

2. Display the Array

3. Sort the Array in Ascending Order by using Max-Heap Sort technique

4. Sort the Array in Descending Order by using any algorithm

5. Time Complexity to sort ascending of random data

6. Time Complexity to sort ascending of data already sorted in ascending order

7. Time Complexity to sort ascending of data already sorted in descending order

8. Time Complexity to sort ascending all Cases (Data Ascending, Data in Descending & Random Data) in Tabular form for values n=5000 to 50000, step=5000

9. Extract largest element

10. Replace value at a node with a new value

11. Insert a new element

12. Delete an element

Enter your choice: 4

Array sorted in descending order: [92, 91, 85, 78, 67, 53, 43, 24, 14, 1]