DATE: 05.12.2023

Assignment 1: Consider a sequence S of n where $n \geq 7$ and implement Heap sort on S.

Theory:

Heap sort is one of the most efficient in-place, unstable sorting algorithm that requires only $O(nlogn)$ operations regardless of the order of the input to sort the sequence. The main idea behind Heap sort is to maintain a Max heap (Min heap in case of descending sorting order) of size n as an almost complete binary tree of n nodes such that the content of each node is less than or equal to the content of its parent node. Now that the root has the maximum value we have to remove it from the heap. Again we have to build the max heap and remove the root element and repeat the procedure until the whole sequence is sorted in ascending order.

Pseudo code:

The pseudo code for heap sort is as follows:

```
Heapsort(a, n)
{ // a[1:n] is the input sequence containing n elements to be sorted using Heap sort

        Heapify(a, n) // Tranform the array into a max heap
                      // Interchange the new maximum with the element at the end of the array

         for i = n to 2
         { // exchanging the value of the root element with the last element
              t = a[i]
              a[i] = a[1]
              a[1] = t
              Adjust(a, 1, i − 1) // Adjusting the tree to maintain the max heap property
         }
}
```

The pseudo code for Heapify function is as follows:

```
Heapify(a, n)
{ // Readjust the elements in a[1:n] to form a heap

         for i = n/2 to 1
         {
              Adjust(a, i, n)
         }
}
```

The pseudo code for Adjust function is as follows:

```
Adjust(a, i, n)
{ // The complete binary trees with roots 2i and 2i + 1 are combined with node i
  // to form a heap rooted at i.
  // No node has an address greater than n or less than 1.

  j = 2i
  item=a[i]
  while (j ≤ n)
  {
       if((j < n) and ( a[j] < a[j + 1]))
       {
            j = j + 1 // Compare left and right child and let j be the bigger child
       }
       if( item ≥ a[j])
```

```
        {
            break from the loop  // A position for item is found
        }
        a[j/2]=a[j]
        j = 2j
    }
    a[j/2]=item

}
```

Assumptions:

The following assumptions have been made during this assignment

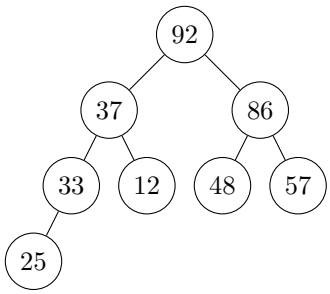
- We are assuming that the given input sequence is of integer type.
- We are assuming that the given sequence is only containing positive integers.

Analysis:

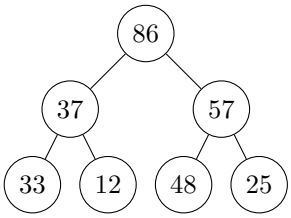
In this approach inside the Heapsort function we have two more functions Heapify to build the max heap and Adjust to adjust the heap after root deletion. Though the call of Heapify requires only $O(n)$ operations, Adjust possibly requires $0(logn)$ operations for each invocation as the worst-case run time of Adjust is also proportional to the height of the tree. Therefore, if there are n elements in a heap, deleting the maximum can be done in $O(logn)$ time. So for n elements it can be done in $O(nlogn)$ time, hence the time complexity for Heap sort is in $O(nlogn)$.

Dry run:

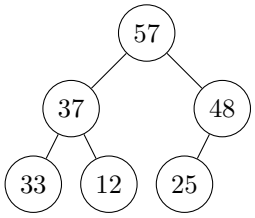
Let us consider the following sequence:[25,57,48,37,12,92,86,33] of size 8, now we will apply heap sort on this sequence to sort it in ascending order.



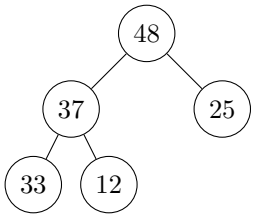
sorted array:[92]



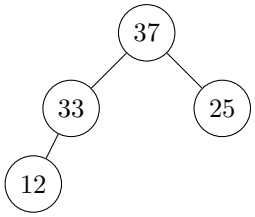
sorted array:[86,92]



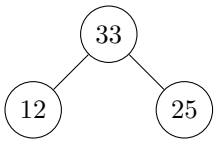
sorted array:[57,86,92]



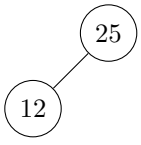
sorted array:[48,57,86,92]



sorted array:[37,48,57,86,92]



sorted array:[33,37,48,57,86,92]



sorted array:[25,37,48,57,86,92]

sorted array:[12,25,37,48,57,86,92]

Code:

Code for Heap sort is as follows:

```
#include <stdio.h>
#include <stdlib.h>

void heapsort(int *, int);    // Funtion prototype for Heap sort
void heapify(int *, int);     // Function prototype for heapify
void adjust(int *, int, int); // Function prototype for adjust which will adjust the heap

int main()
{
    int *a, n;
    do
    {
        printf("How many elements to be inserted?:  ");
        scanf("%d", &n);
        if (n <= 0)
        {
            printf("\nEnter a valid number!!!..\n");
        }

    } while (n <= 0);

    // Dynamic memory allocation for input array
    a = (int *)calloc(n, sizeof(int));
    printf("Enter the elements:\t");
    for (int i = 0; i < n; i++)
    {

        scanf("%d", &a[i]);
    }

    heapsort(a, n - 1); // Calling the function for heap sort
    printf("\n\narray after heapsort:\n\n");
    for (int i = 0; i < n; i++)
    { // printing the sorted array

        printf("\t%d", a[i]);
    }
    printf("\n");
    free(a); // Free the input array
    return 0;
}

void heapsort(int *a, int n)
{ // Function definition for heapsort

    heapify(a, n); // Calling the function to build the initial max heap

    for (int i = n; i >= 1; i--)
    { // placing the root value in it's appropriate position
        int t = a[i];
        a[i] = a[0];
        a[0] = t;
        adjust(a, 0, i - 1); // Calling the function to adjust the heap
    }
}

void heapify(int *a, int n)
{ // Function definition to build the heap

    for (int i = n / 2; i >= 0; i--)
    {
        adjust(a, i, n);
    }
}
```

```
void adjust(int *a, int i, int n)
{ // Function definition to adjust the heap
    int j = 2 * i;
    int item = a[i];
    while (j <= n)
    {
        if ((j < n) && (a[j] < a[j + 1]))
        { // Compare left and right child and let j be the bigger child
            j = j + 1;
        }
        if (item >= a[j])
        {
            break; // A position for item is found
        }
        a[j / 2] = a[j];
        j = 2 * j;
    }
    a[j / 2] = item;
}
```

Output:

```
How many elements to be inserted?:  8
Enter the elements:     25 57 48 37 12 92 86 33


array after heapsort:

        12      25      33      37      48      57      86      92



How many elements to be inserted?:  10
Enter the elements:     12 4 99 23 3 54 81 77 36 17


array after heapsort:

        3       4       12      17      23      36      54      77      81      99
```

Discussion:

- Heap sort has a time complexity of $O(nlogn)$ where n is the number of elements to be sorted for all the three cases ;.e. best,worst,average case .
- Heap sort invloves building and maintaining heap which makes this algorithm very costly.
- Heap sort is an unstable algorithm which means it can change the relative order of equal terms.