

# DAA Lab-7

Name: Kshitij Kumar Sharma

Roll No.: 1905514

Date: 02/09/2021

---

**Q-7.1)** Write a program to sort a given set of elements with the Heap sort.

- 1) Repeat the experiment for different values of  $n = 20000, 50000, 100000, 500000$  and report the time (in seconds) required to sort the elements.
- 2) For each of aforementioned case, consider arrays as random, sorted, and reverse-sorted and observe running time variation for different types of input for heap sort. [Provide your observation regarding sensitivity of heap sort on the input in your lab record.]

**Program:**

/\*

Written by: Kshitij Kumar Sharma

Roll No.: 1905514

Idea of the solution:

I have implemented heapify() to convert the array into a heap data structure and generate the max heap. The heap\_sort() function calls the heapify(). The idea behind the concept is that the parent node is always greater than the child nodes and that is what I have tried to implement through this program.

\*/

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
void heapify(int a[], int n, int i)
```

```
{
```

```
    int l,r,largest=i;
```

```
    l=2*i+1;
```

```
    r=2*i+2;
```

```
    if (l<n&& a[l]>a[largest])
```

```
        largest=l;
```

```
    if (r<n&& a[r]>a[largest])
```

```
        largest=r;
```

```
    if (largest!=i)
```

```
    {
```

```
        swap(a[i],a[largest]);
```

```
        heapify(a,n,largest);
```

```
    }
```

```
}
```

```
void heap_sort(int a[], int n)
```

```
{
```

```
    int i;
```

```
    for (i=n/2-1;i>=0;i--)
```

```
        heapify(a,n,i);
```

```
    for (i=n-1;i>0;i--)
```

```
    {
```

```
        swap(a[0],a[i]);
```

```
//converting to heap data structure
```

```
//and max-heap generation
```

```
//left child
```

```
//right child
```

```
//checking for the largest among the three
```

```
//swapping
```

```
//recursively calling
```

```
//heap sort
```

```

        heapify(a,i,0);
    }
}
int main()
{
    int n,i,j,k,s[]={20000,50000,100000,500000}; //different array sizes to test
    clock_t start, end; //time variables for timing analysis
    double cpu_time_used;
    for(i=0;i<4;i++)
    {
        cout<<endl;
        n=s[i];
        int a[n];
        srand(time(0));
        for(j=0;j<n;j++)
            a[j]=rand()%1000000; //generating random array
        cout<<"For n= "<<n<<endl;
        start=clock();
        heap_sort(a,n);
        end=clock();
        cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;
        printf("Time taken for random : %fsec \n",cpu_time_used);

        sort(a,a+n);

        start=clock();
        heap_sort(a,n);
        end=clock();

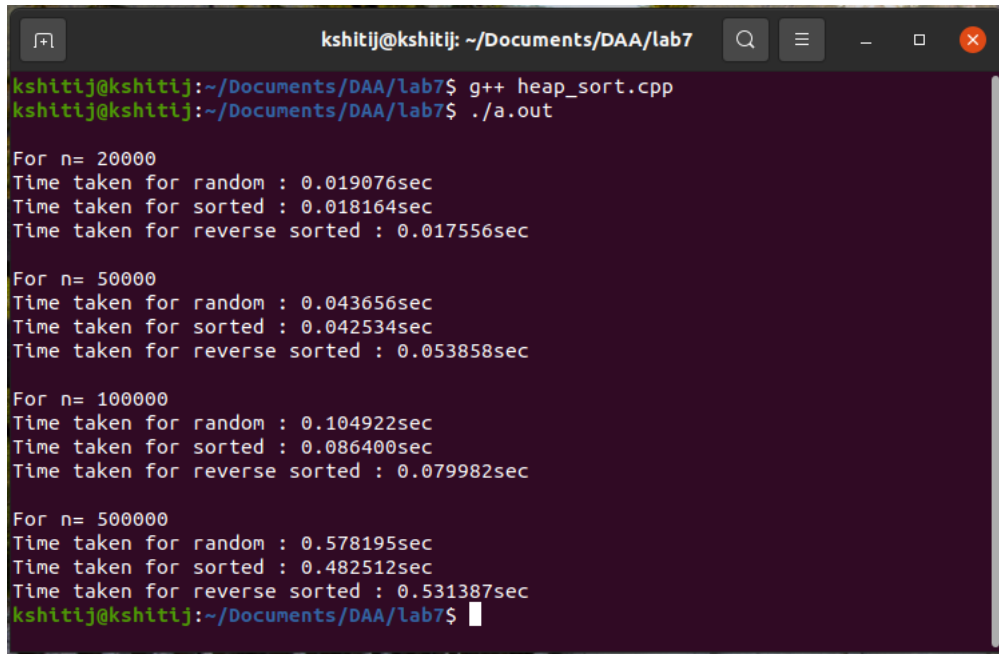
        cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;
        printf("Time taken for sorted : %fsec \n",cpu_time_used);

        sort(a,a+n,greater<int>());

        start=clock();
        heap_sort(a,n);
        end=clock();
        cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;
        printf("Time taken for reverse sorted : %fsec \n",cpu_time_used);
    }
}

```

## Output:



```
kshitij@kshitij: ~/Documents/DAA/lab7
kshitij@kshitij:~/Documents/DAA/lab7$ g++ heap_sort.cpp
kshitij@kshitij:~/Documents/DAA/lab7$ ./a.out

For n= 20000
Time taken for random : 0.019076sec
Time taken for sorted : 0.018164sec
Time taken for reverse sorted : 0.017556sec

For n= 50000
Time taken for random : 0.043656sec
Time taken for sorted : 0.042534sec
Time taken for reverse sorted : 0.053858sec

For n= 100000
Time taken for random : 0.104922sec
Time taken for sorted : 0.086400sec
Time taken for reverse sorted : 0.079982sec

For n= 500000
Time taken for random : 0.578195sec
Time taken for sorted : 0.482512sec
Time taken for reverse sorted : 0.531387sec
kshitij@kshitij:~/Documents/DAA/lab7$
```

**Q-7.2)** Compare running time (in seconds) of heap sort with insertion sort, merge sort, and quick sort on different input sizes and also for different input types (random/sorted/reverse-sorted).

### **Program:**

/\*

Written by: Kshitij Kumar Sharma

Roll No.: 1905514

Idea of the solution:

The concept of heap sort is same just adding merge sort and random quick sort which I have did in previous labs. Included them as a header file in this program to avoid repetition of the same code.

\*/

```
#include <bits/stdc++.h>
```

```
#include "merge_sort.cpp"
```

```
//merge sort header file
```

```
#include "quick_sort_random.cpp"
```

```
//random quick sort header file
```

```
using namespace std;
```

```
//everything remains same as before
```

```
void heapify(int a[], int n, int i)
```

```
{
```

```
    int l,r,largest=i;
```

```
    l=2*i+1;
```

```
    r=2*i+2;
```

```
    if (l<n&&a[l]>a[largest])
```

```

        largest=l;
    if (r<n&& a[r]>a[largest])
        largest=r;
    if (largest!=i)
    {
        swap(a[i],a[largest]);
        heapify(a,n,largest);
    }
}

void heap_sort(int a[], int n)
{
    int i;
    for (i=n/2-1;i>=0;i--)
        heapify(a,n,i);
    for (i=n-1;i>0;i--)
    {
        swap(a[0],a[i]);
        heapify(a,i,0);
    }
}

int main()          //some changes into the main function for analysing all the algorithms together
{
    int n,i,j,k,s[]={20000,50000,100000,500000};
    clock_t start, end;
    double cpu_time_used;
    for(i=0;i<4;i++)
    {
        cout<<endl;
        n=s[i];
        int a[n];
        srand(time(0));
        for(j=0;j<n;j++)
            a[j]=rand()%1000000;
        cout<<"For n= "<<n<<endl;
        cout<<" Random Array \n";
        start=clock();
        heap_sort(a,n);
        end=clock();
        cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;
        printf(" Heap sort : %fsec \n",cpu_time_used);

        start=clock();
        merge_sort(a,0,n-1);
        end=clock();
    }
}

```

```
cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;  
printf(" Merge sort : %fsec \n",cpu_time_used);
```

```
start=clock();  
random_quick_sort(a,0,n-1);  
end=clock();  
cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;  
printf(" Quick sort : %fsec \n",cpu_time_used);  
cout<<" Sorted Array \n";  
sort(a,a+n);
```

```
start=clock();  
heap_sort(a,n);  
end=clock();  
cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;  
printf(" Heap sort : %fsec \n",cpu_time_used);
```

```
start=clock();  
merge_sort(a,0,n-1);  
end=clock();  
cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;  
printf(" Merge sort : %fsec \n",cpu_time_used);
```

```
start=clock();  
random_quick_sort(a,0,n-1);  
end=clock();  
cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;  
printf(" Quick sort : %fsec \n",cpu_time_used);  
cout<<" Reverse Sorted Array \n";  
sort(a,a+n,greater<int>());
```

```
start=clock();  
heap_sort(a,n);  
end=clock();  
cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;  
printf(" Heap sort : %fsec \n",cpu_time_used);
```

```
start=clock();  
merge_sort(a,0,n-1);  
end=clock();  
cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;  
printf(" Merge sort : %fsec \n",cpu_time_used);
```

```
start=clock();
```

```

random_quick_sort(a,0,n-1);
end=clock();
cpu_time_used=((double)(end-start))/CLOCKS_PER_SEC;
printf(" Quick sort : %fsec \n",cpu_time_used);
}
}

```

## Output:

```

kshiti@kshiti: ~/Documents/DAA/lab7
kshiti@kshiti:~/Documents/DAA/lab7$ g++ heap_sort_compare.cpp
kshiti@kshiti:~/Documents/DAA/lab7$ ./a.out

For n= 20000
Random Array
Heap sort : 0.019839sec
Merge sort : 0.004904sec
Quick sort : 0.046127sec
Sorted Array
Heap sort : 0.013657sec
Merge sort : 0.003371sec
Quick sort : 0.049512sec
Reverse Sorted Array
Heap sort : 0.019745sec
Merge sort : 0.005941sec
Quick sort : 0.044134sec

For n= 50000
Random Array
Heap sort : 0.053084sec
Merge sort : 0.012352sec
Quick sort : 0.105423sec
Sorted Array
Heap sort : 0.041788sec
Merge sort : 0.010846sec
Quick sort : 0.101728sec
Reverse Sorted Array
Heap sort : 0.039828sec
Merge sort : 0.009499sec
Quick sort : 0.104076sec

For n= 100000
Random Array
Heap sort : 0.095071sec
Merge sort : 0.022832sec
Quick sort : 0.201117sec
Sorted Array
Heap sort : 0.097461sec
Merge sort : 0.022608sec
Quick sort : 0.175769sec
Reverse Sorted Array
Heap sort : 0.080033sec
Merge sort : 0.022906sec
Quick sort : 0.164138sec

For n= 500000
Random Array
Heap sort : 0.607102sec
Merge sort : 0.135853sec
Quick sort : 1.059321sec
Sorted Array
Heap sort : 0.504599sec
Merge sort : 0.128213sec
Quick sort : 1.043590sec
Reverse Sorted Array
Heap sort : 0.505018sec
Merge sort : 0.162215sec
Quick sort : 1.011279sec
kshiti@kshiti:~/Documents/DAA/lab7$

```