Lab 7 (Heap Sort)

1.0 Time Complexity

Heap Sort takes time for

- **Build-heap**
- **Heapify n times**

Build-heap time taken can be represented as **O(n/2)**; Because build-heap runs only n/2 times.

Heapify time taken can be represented as **O(logn)**; Because heapify is going through one branch from each of two branches in the binary tree structure. This process runs **n** times until all the elements are removed from the array and arranged as a sorted array.

After considering both cases we can say that,

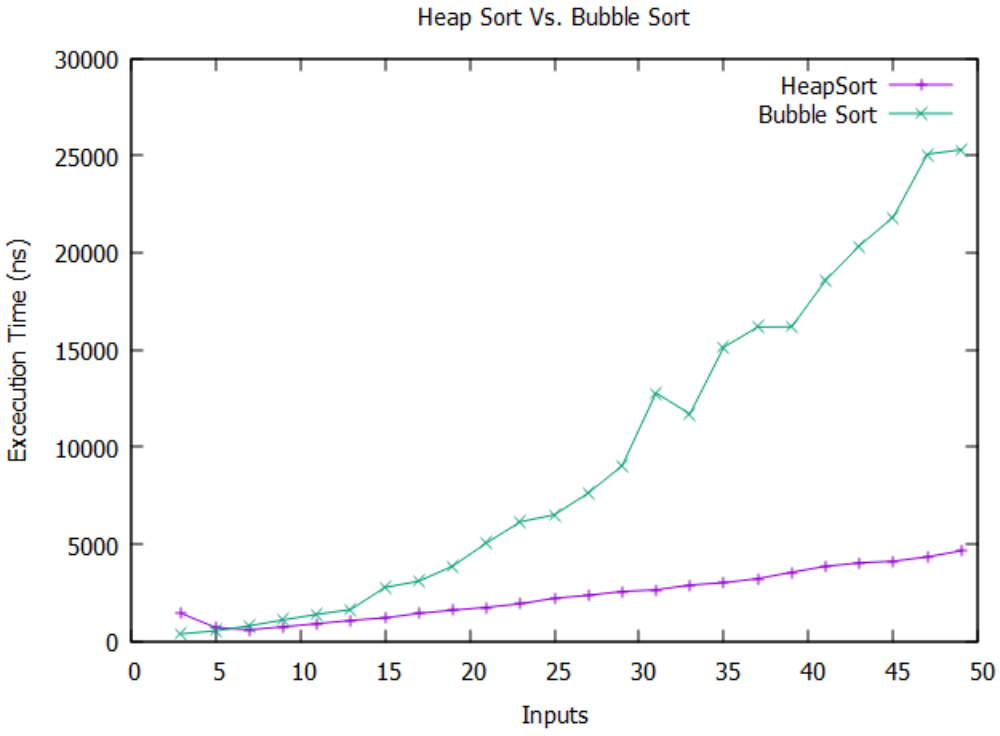
T(n) = C1*(n/2) + C2*n*logn + n*C3 ; C3*n for exchanging elements inside the same loop with heapify

So we can say that the time complexity of heap sort is O(nlogn)

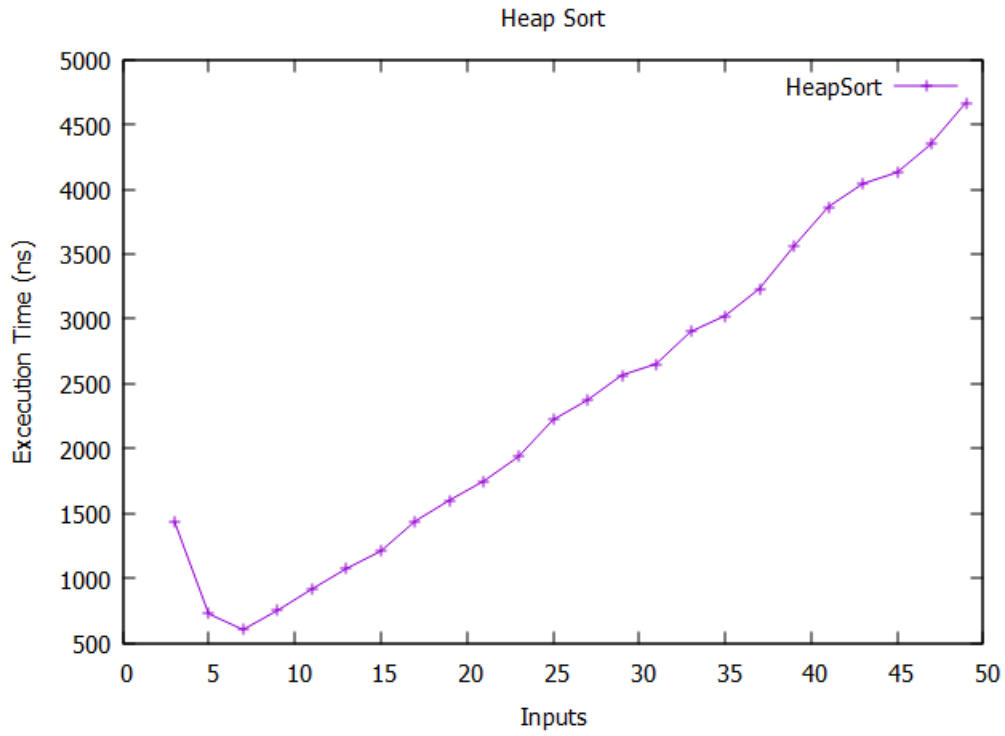
Since that,

It should be faster than Bubble sort, Insertion sort,... and slower than Merge Sort.

2.0 Graph (Heap vs Bubble)



3.0 Graph (HeapSort)



4.0 Code for Graph

```
#include <iostream>
using namespace std;
int main() {
  FILE *gnuplotPipe = popen("gnuplot -persistent", "w");
  fprintf(gnuplotPipe, "set title 'Heap Sort'\n");
  fprintf(gnuplotPipe, "set xlabel 'Inputs'\n");
  fprintf(gnuplotPipe, "set ylabel 'Excecution Time (ns)'\n");
  fprintf(gnuplotPipe, "plot 'data1.txt' with linespoints title 'HeapSort'\n");
  fflush(gnuplotPipe);
  return 0;
}
```

5.0 Data

```
HeapSort
#########
3   1442.5999999999990905053
5   727.39999999999997726263
7   605.20000000000004547474
9   753.39999999999997726263
11  917.79999999999995452526
```

```
13  1076.20000000000004547474
15  1210.40000000000009094947
17  1440.79999999999995452526
19  1602.79999999999995452526
21  1751.20000000000004547474
23  1937.79999999999995452526
25  2222.19999999999981810106
27  2374.40000000000009094947
29  2566.40000000000009094947
31  2653.19999999999981810106
33  2901.40000000000009094947
35  3023.59999999999990905053
37  3231.80000000000018189894
39  3560.59999999999990905053
41  3863.19999999999981810106
43  4041.19999999999981810106
45  4129.60000000000036379788
47  4350.19999999999981810106
49  4660.80000000000018189894
Bubble Sort
#########
3   384.60000000000002273737
5   545.20000000000004547474
7   801.60000000000002273737
9   1102.20000000000004547474
11  1392.59999999999990905053
13  1635.20000000000004547474
15  2779.19999999999981810106
17  3095.80000000000018189894
19  3873.19999999999981810106
21  5065.00000000000000000000
23  6155.60000000000036379788
25  6510.39999999999963620212
27  7624.39999999999963620212
29  9027.20000000000072759576
31  12781.79999999999927240424
33  11698.20000000000072759576
35  15132.39999999999963620212
37  16176.20000000000072759576
39  16178.20000000000072759576
41  18560.59999999999854480848
43  20360.40000000000145519152
45  21831.00000000000000000000
47  25055.00000000000000000000
49  25297.40000000000145519152
```

6.0 Code for Data

```
#include <iostream>
#include <vector>
#include <chrono>
using namespace std;
void print(int n,vector<int> arr)
{
    for(int i=0; i<n; i++) {
        std::cout<<arr[i]<<" ";
    }
    std::cout<<"\n";
}
vector<vector<int>> makeRandomArrays(int start_size,int end_size,int step, int
value_limit)
{
    vector<vector<int>> arrays;
    vector<int> sample;
    for(int i=start_size; i<end_size+1; i=i+step) {
        sample.clear();
        for(int j=0; j<i; j++) {
            sample.push_back(rand()%(value_limit+1));
        }
        arrays.push_back(sample);
    }
    return arrays;
}

//Add your programs and other functions here.
// function to heapify the tree
void heapify(int arr[], int n, int root)
{
    // build heapify
    int left=2*root+1;
    int right=2*root+2;
    int maximum;
    if(left<n && arr[left]>arr[root]) {
        maximum=left;
    } else {
        maximum=root;
    }
    if(right<n && arr[right]>arr[maximum]) {
        maximum=right;
    }
    if(maximum!=root) {
```

```
        int temp=arr[root];
        arr[root]=arr[maximum];
        arr[maximum]=temp;
        heapify(arr,n,maximum);
    }


}

// implementing heap sort
void heapSort(int arr[], int n)
{
    // build heap
    for(int i=n/2-1; i>=0; i--) {
        heapify(arr,n,i);
    }


    // extracting elements from heap one by one
    while(n>0) {
        int temp=arr[n-1];
        arr[n-1]=arr[0];
        arr[0]=temp;
        n--;
        heapify(arr,n,0);
    }
}

void swap(int &a,int &b)
{
    int c=a;
    a=b;
    b=c;
}
void bubbleSort(int n,vector<int> &array)
{
    for(int i=0; i<n; i++) {
        for(int j=0; j<n-1; j++) {
            if(array[j]>array[j+1]) { //check wheather the next value is greater
than current value
                swap(array[j+1],array[j]);// swap values
            }
        }
    }
}
```

```
void runtheProgram1(int n,vector<int> inputs)
{
    int arr[n];
    for(int i=0; i<n; i++) {
        arr[i]=inputs[i];
    }
    heapSort(arr,n);
}
void runtheProgram2(int n,vector<int> inputs)
{
    bubbleSort(n,inputs);

}

int main()
{
    //Get the values
    vector<vector<int>> arrays=makeRandomArrays(3,50,2,100);

    double sum_duration;
    vector<double> avg_duration;
    string topic;
    for(int sorting=0; sorting<2; sorting++) { // change the number of sorting
algorithms (sorting < (number of algorithms))
        avg_duration.clear();
        for(int t=0; t<arrays.size(); t++) {
            sum_duration=0.0f;

            for(int i=0; i<5; i++) { //5 times

                auto start = chrono::high_resolution_clock::now();


                switch(sorting) {
                case 0:
                    runtheProgram1(arrays[t].size(),arrays[t]);
                    topic="\n\n\nHeapSort\n#########\n";
                    break;
                case 1:
                    runtheProgram2(arrays[t].size(),arrays[t]);
                    topic="\n\n\nBubble Sort\n#########\n";
                    break;
                default:
                    break;
                }
```

```
                auto end = chrono::high_resolution_clock::now();

                // Calculating total time taken by the program.
                double time_taken =
                    chrono::duration_cast<chrono::nanoseconds>(end -
start).count();


                sum_duration=sum_duration+time_taken;


            }
            avg_duration.push_back(sum_duration/5.0f);
        }
        cout<<topic;
        for(int i=0; i<avg_duration.size(); i++) {
            printf("%i\t%.20f\n",i*2+3,avg_duration[i]);
        }
    }
    return 0;
}
```