Extraction of maximum and minimum element during each iteration of Heapsort Algorithm

Presented By:-
Maharshi Roy (ISM2014006)
Rajat Kumar Sahu (IHM2014501)
Amrit Daimary (IRM2014001)

Extraction of maximum and minimum element

- We present a methodology to extract both maximum and minimum element during each iteration of Heapsort Algorithm. So during each iteration the size of heap reduces by 2 instead of 1.

- First we build a max heap out of the input array.

- We extract the maximum element that is the root element and place it in the output array and replace the root with the last element of the heap.

- Then we reduce the heap size by 1 and call maxheapify.

- Since the smallest element belong to leave nodes we iterate over all the leave nodes and extract the last element to the output array and place the last element of the heap in it's place. Then we reduce the heap size by 1 and call the float up function to maintain the heap property.

Algorithm

```
floatup (A[], n, index) {
    parent=index/2;
    while(parent>=1 and A[parent]<arr[index]){
       swap(A[index],A[parent])


    index=parent
    parent=index/2;
}
```

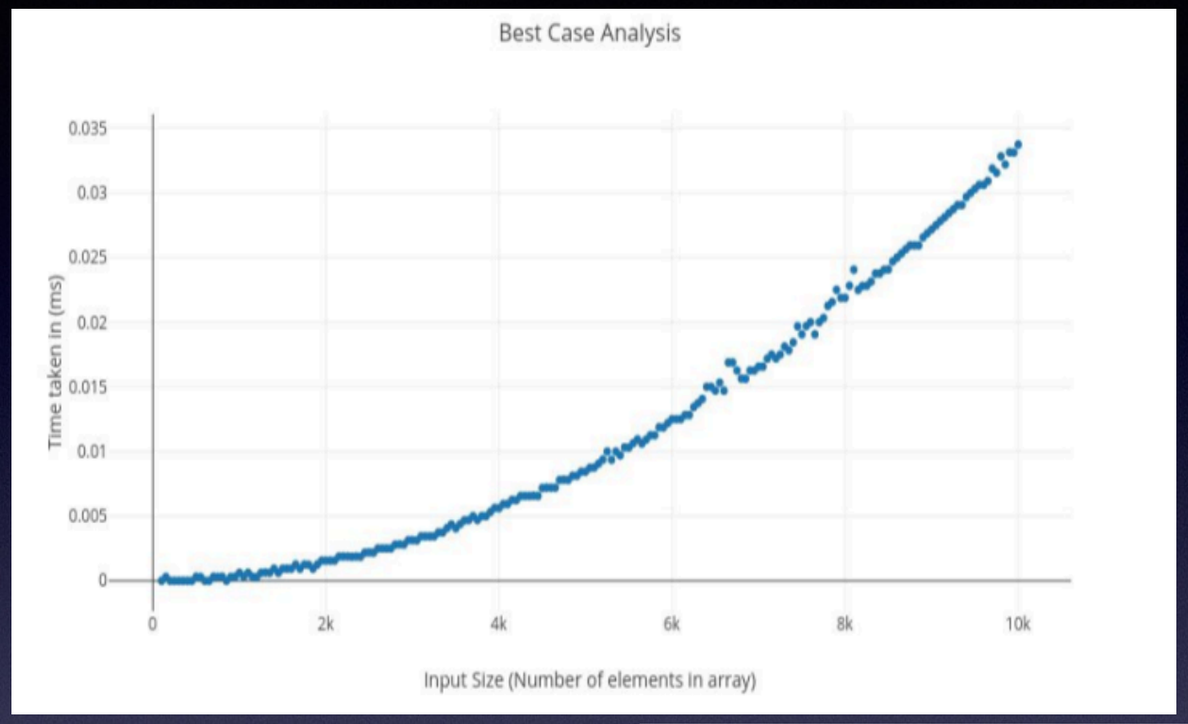
Algorithm

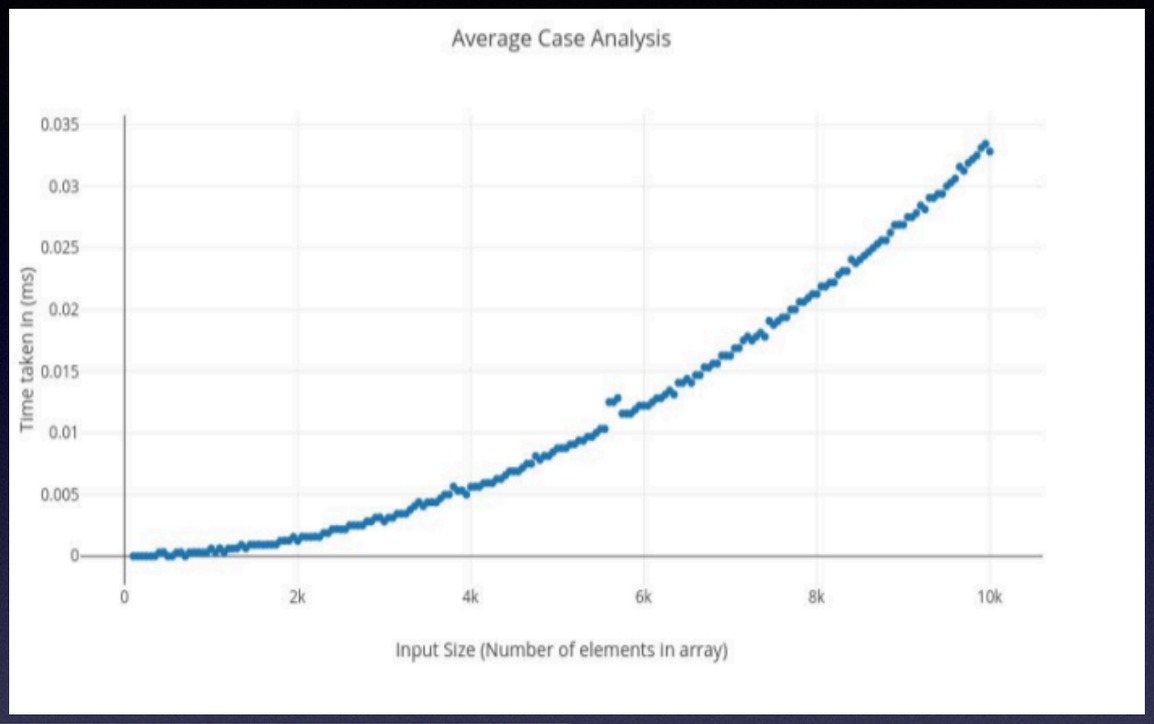
```
maxheapify (A[], n, index) {
        largest=index
        left=2*index
        right=left+1

        if (left<=n && A[left] > A[largest]) {
                largest=left
        }

        if (right<=n && A[right] > A[largest]) {
                largest=right
        }

        if (index!=largest) {
                swap(A[index],A[largest])
                maxheapify(A,n,largest);
        }
}
```

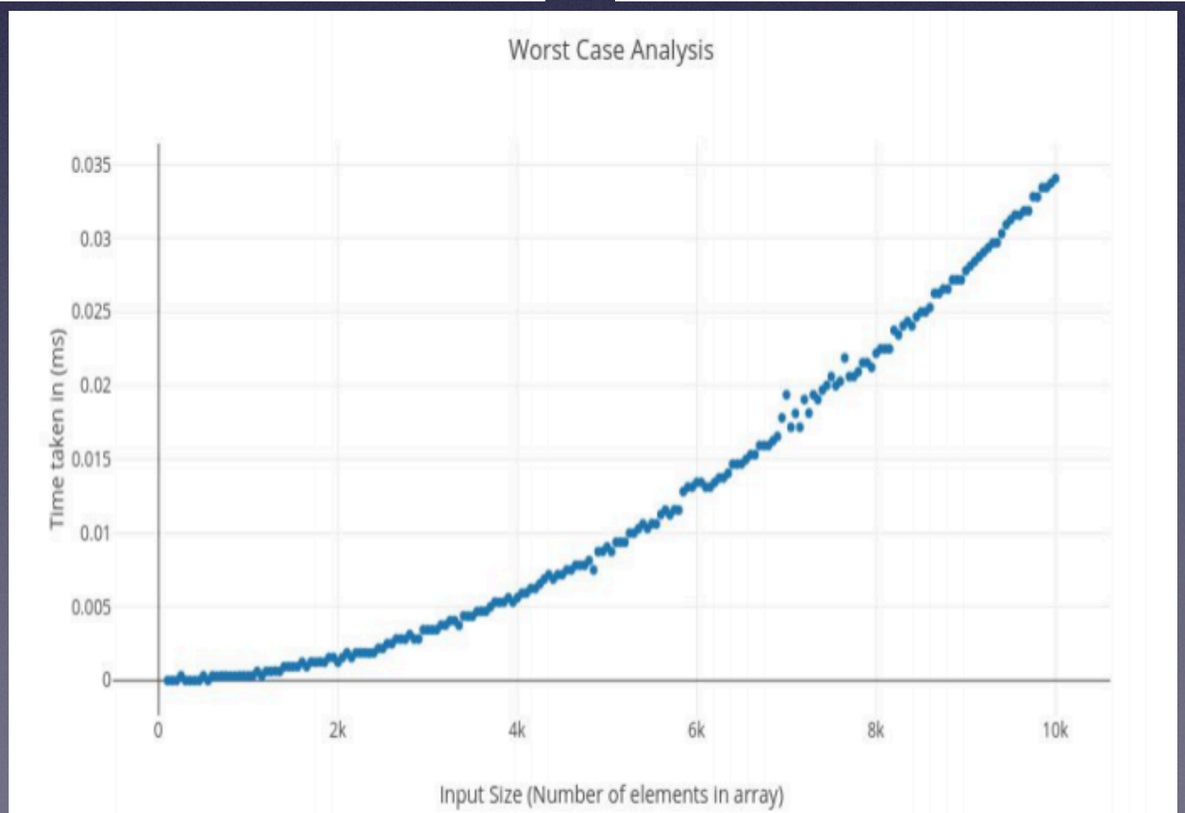
Algorithm

```
heapsort (A[], O[], n) {
        k=1
        for (i from n/2 to 1) {
                maxheapify(A,n,i)
        }
        for (i from n to 1) {
                O[k]=A[1]
                A[1]=A[i]
                i=i-1
                maxheapify(A,i,1)
                if (i<1)
                    break
                small=i/2+1
                for (j from i/2 + 2 to i) {
                        if(arr[small] > arr[j]) {
                                small=j;
                        }
                }
                O[n-k+1]=A[small]
                k=k+1
                A[small]=A[i]
                i=i-1
                floatup(A,small);
        }
}
```

COMPLEXITY ANALYSIS







THANK YOU