Importance of Effecient Sorting
Featuring Heap Sort

Joshua Pondrom

October 2, 2017

1 Motivation and History

Heap Sort is a solution to a famous Computer Science problem, sorting. Sorting is problem which is defined as follows:

Given a list of elements,

$$a_1, a_2, a_3, \ldots, a_{n-1}, a_n$$

rearrange the list such that

$$a_1 < a_2 < a_3 < \ldots < a_{n-1} < a_n$$

People have been working on the design and creation of better sorting algorithms since the beginning of computing, it is a problem which can apply to many facets of the world of computing. There are enough popular sorting algorithms to where they started to be categorized into how the solved the problem, such as selection, insertion, partitioning, merging, and more. Heap Sort is one of the sorting algorithms which uses selection to solve the problem and also makes use of the heap data structure.

The heap data structure is a tree which is complete until its last level of depth, and any parent node will be greater than both of its children. In general, the heap can help solve the sorting problem because heap can find a maximum element in constant time, $O(1)$, and remove and re-heapify in logarithmic time, $O(log(n))$. This is helpful because the algorithm can store every element in a heap, get the maximum element then remove it from the heap, then continue to get the maximum of the remaining elements until it finishes with no elements left in the heap, and a fully sorted list.

2 Background and Comparisons

As mentioned previously, there are plenty of ways to tackle the sorting problem, so how do other methods solve the problem that differ from Heap Sort? Lets look at other solutions to the problem such as selection sort, merge sort, and bubble sort.

Bubble sort is a very simple and basic sorting algorithm that most computer scientists will know of. It is a very intuitive approach to the sorting problem. The way it works is it will go through the list for n amount of times and switch the current element it is looking at and the one proceeding it if it happens to be less than it. However this is a very inefficient algorithm which has best and average case runtime $O(n^2)$ whereas heap sort runs in $O(nlog(n))$.

Merge sort is an a merging based algorithm which competes with Heap sort in terms of algorithmic complexity. It's worst, best, and average cases all run at $O(nlog(n))$, making it a very consistent algorithm, like Heap sort. The basis of Merge sort is the concept behind dynamic programming, it breaks the problem into subproblems and works its way up. In the case of sorting, it breaks the array up as far as possible, and merges the two of the sections together at a time, making comparisons between two sections. Repeat this step ad nauseum until the algorithm finishes with a sorted array.

Lastly, we have Selection sort, another well known beginner algorithm like bubble sort. Like bubble sort, this algorithms worst, best, and average case are $O(n^2)$. This algorithm uses the selection method to solve the problem. The way this works is it will create a extra list, do a normal search through the list of elements to find the lowest, then put the lowest in the extra list. It will repeat this until the list of elements is empty and the extra list has all of the elements in sorted order. This algorithm is important because while it has a subpar algorithmic complexity, it is the foundation of heap sort. Heap sort solves the sorting problem in the same method as selection sort, but heap sort improves on it through the use heap data structure.

3 Pseudocode and Testing

3.1 Bubble Sort

```
procedure BubbleSort(List:L)
        n = L.length
        for(i = 0 to n-1)
          for(j = 0 to n-i-1)
                  if(L[j] > L[j+1])
                          swap(L[j],L[j+1])
```

3.2 Merge Sort

```
procedure MergeSort(List:L, p , r)
        if p < r
                q = floor((p+r)/2)
                MergeSort(L, p, q)
                MergeSort(L, q + 1, r)
                Merge(A, p, q, r)
                #Merge is a function which combines
                #the two sublists into one sorted
                #list containing all of the elements
                #of the two parent lists
```

3.3 Heap Sort

```
procedure HeapSort(List:L)
        BuildMaxHeap(L)
        for(i = L.length to 2)
                swap(L[1], L[i])
                L.size = L.size - 1
                MaxHeapify(L,1)

procedure BuildMaxHeap(L)
        L.size = L.length
        for(i = floor(L.length/2) to 2)
                MaxHeapify(L,i)

procedure MaxHeapify(L,i)
        left = 2*i
        right = 2*i + 1
        if(left <= L.size and A[left] > A[i]
                greatest = left
        else
                greatest = i
        if(right <= L.size and A[right] > A[largest]
                greatest = right
        if(greatest != i)
                swap(L[i],L[greatest]
                MaxHeapify(L, greatest)
```

4 Testing and Conclusion

4.1 Implementation

Currently, only pseudocode was mentioned for Heap sort, but how would it end up implemented? First, a language must be chosen. Python would be a good fit for testing the algorithm, it is simple to implement the algorithm without worrying about low level computing such as memory management and type errors. Additionally, Python contains some helpful libraries such as 'matplotlib' and 'seaborn' to help visualize the data to check for correctness and to show the sorting algorithm working step-by-step. Additionally we can look at the data by using simple standard console I/O.

4.2 Mathematical Runtime

From the algorithmic complexity given earlier, we know the runtime will always be $O(nlog(n))$ at best case, worst case, and average case. That means that for any given amount of elements, the order does not matter to how well the algorithm will perform. We can calculate how well the algorithm performs versus some of the other algorithms in terms of values of n and the amount of steps performed.

n	$nlog(n)$	n^2
100	664	10000
500	4483	250000
750	7163	526500
1000	9966	1000000

As shown by the table, algorithms complexity analysis is a must. The Heap sort is sorting at a rate of 1000 times less operations than a simple selection sort or bubble sort, and this rate will only go further in the favor of Heap sort as the number of elements increases.