Aneesh Abhyankar
RUID - 166006555
Net ID – ana85

Assignment 3

1) View the following Data Set here. The column on the left is the original input of strings to be sorted or shuffled; the column on the extreme right are the string in sorted order; the other columns are the contents at some intermediate step during one of the 8 algorithms listed below. Match up each algorithm under the corresponding column. Use each algorithm exactly once: (1) Knuth shuffle (2) Selection sort (3) Insertion sort (4) Merge sort(top-down) (5) Merge sort (bottom-up) (6) Quicksort (standard, no shuffle) (7) Quicksort (3-way, no shuffle) (8) Heapsort.

1. Bottom Up Merge Sort

   Pairs of four elements are sorted among themselves, similar to what happens in bottom up merge sort.

2. Quick Sort (standard, no shuffle)

   First element (navy) becomes the pivot and all elements before ‘navy’ are smaller and all those after ‘navy’ are larger. First step that takes place here is that ‘mist’ and ‘plum’ get swapped.

3. Knuth Shuffle

   Elements are placed at random and do not follow a particular logic.

4. Top Down Merge Sort

   First half and the second half are individually sorted and elements are not yet in their final positions.

5. Insertion Sort

   Second half of the array is exactly same as the original, because it has not been seen yet, and the first half is sorted.

6. Heapsort

   The element at position 1 is greater than elements at positions 2 and 3. Element at position 2 is greater than elements at positions 4 and 5 and so on. In general, element at position ‘n’ is greater than elements at positions ‘2n’ and ‘2n + 1’, because it is their parent. This is an example of max-heap.

7. Selection Sort

   First half of the array is exactly same as the sorted array, because all the elements in the first half have been selectively placed in their final positions.

8. Quick Sort (3-way, no shuffle)
First element (navy) becomes the pivot and all elements before ‘navy’ are smaller and all those after ‘navy’ are larger. First step that takes place here is that ‘plum’ and ‘palm’ get swapped and plum ends up in the last position

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

2) Implement Quicksort using median-of-three to determine the partition element. Compare the performance of Quicksort with the Merge sort implementation and dataset from Q1 (HW2). Is there any noticeable difference when you use N=7 as the cut-off to insertion sort? Experiment if there is any value of "cut-off to insertion" at which the performance inverts.

The table below shows the comparative time performance of Median of Three Quick Sort and Quick Sort that reverts to Insertion Sort below a certain length given by cutoff value with that of Top Down Merge Sort. The graph below the table shows the data below plotted against data size.

Data Size	Execution Time (ms)			
	Top-Down Merge-sort	Median of Three Quicksort	Median of Three Quicksort (Cutoff=7)	Median of Three Quicksort (Cutoff=5)
1024	16	7	8	7
2048	40	14	17	14
4096	91	36	36	35
8192	196	79	77	60
16384	429	175	174	150
32768	1003	333	348	315

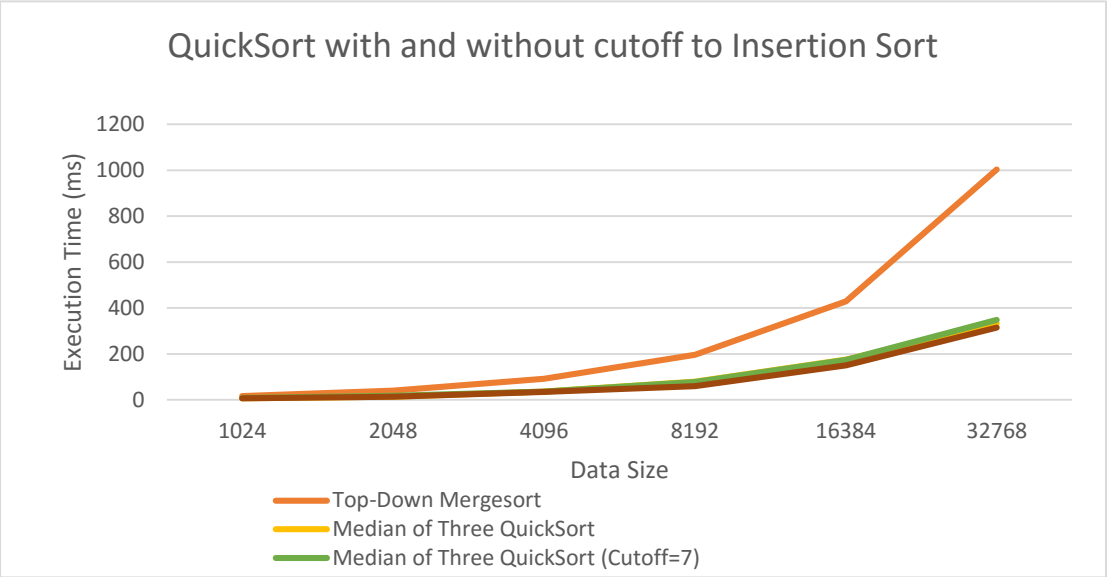

Figure 1 – Performance time comparison of two Quick Sorts with Merge Sort

We see that the time taken for execution increases with data size as expected. The quick sort with median-of -three algorithm out performs merge sort. The order of growth of quick sort is also less than that of merge sort. The time insertion sort takes to sort small sized arrays is less than quick sort. So, cutting off to insertion sort for small sub-arrays pays off as seen from the graph. The trend, however, inverts above the cutoff value of 25.

Hypothesis => for Merge sort

$T = a*N^b$

Therefore, $40 = a*2048^b$ and $91 = a*4096^b$

Solving, we get b = 1.186 and a = 0.00473 (approximately)

Thus, $\mathbf{T = 0.00473*N^{1.186}}$

Hypothesis => for Median-of-Three Quick sort

$T = a*N^b$

Therefore, $14 = a*2048^b$ and $36 = a*4096^b$

Solving, we get b = 1.345 and a = 0.00049 (approximately)

Thus, $\mathbf{T = 0.00049*N^{1.345}}$

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Aneesh Abhyankar
RUID - 166006555
Net ID – ana85

3) Problem 9.29 From Sedgewick, Algorithms in C++, 3rd Edition: Empirically determine the percentage of time heapsort spends in the construction phase for N=10^3, 10^4, 10^5, and 10^6.

The table below shows time taken by Heap Sort in construction phase of the algorithm, the total time taken by the algorithm and subsequently, the percentage of time spent in construction phase. Figure 2 plots heap construction time with respect to data size and Figure 3 plots the percent time spent in construction phase against data size. The data size considered in the analysis is 10^3, 10^4, 10^5, and 10^6.

Data Size	Time in Construction	Total Time	% Time in Construction
1000	1	17	5.882352941
10000	20	236	8.474576271
100000	171	2211	7.734056988
1000000	1333	26095	5.108258287

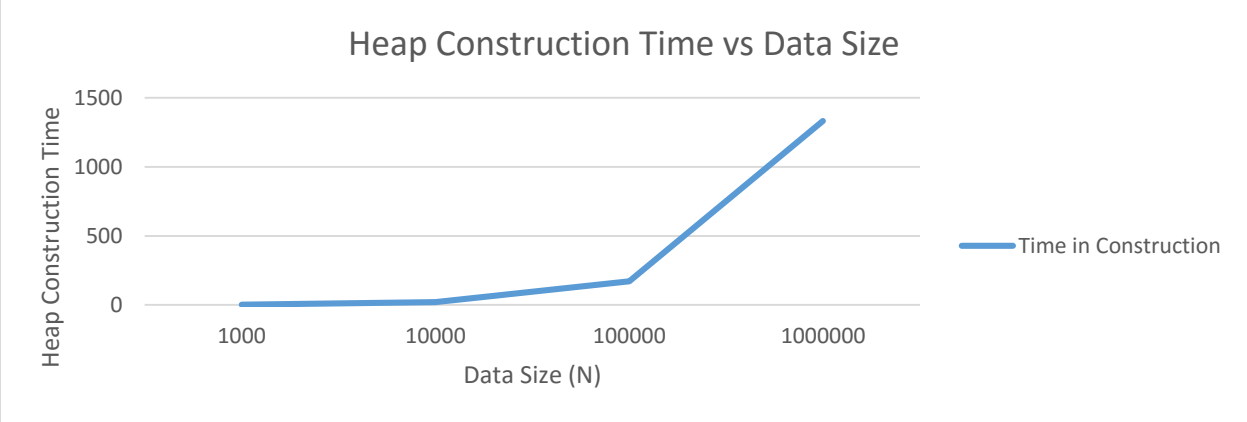

Figure 2 – Heap Construction time vs Data Size

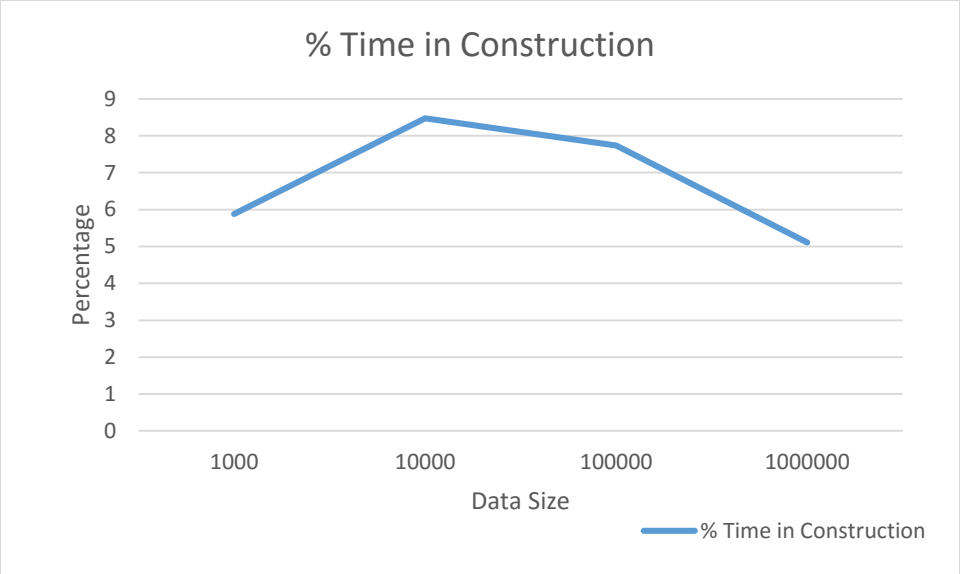

Figure 3 – Percent time spent in construction phase vs data size

Hypothesis => for Time spent in construction phase by heap sort

$T = a*N^b$

Therefore, $18 = a*10000^b$ and $175 = a*100000^b$

Solving, we get b = 0.987 and a = 0.00203 (approximately)

Thus, $\mathbf{T = 0.00203*N^{0.987}}$

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx