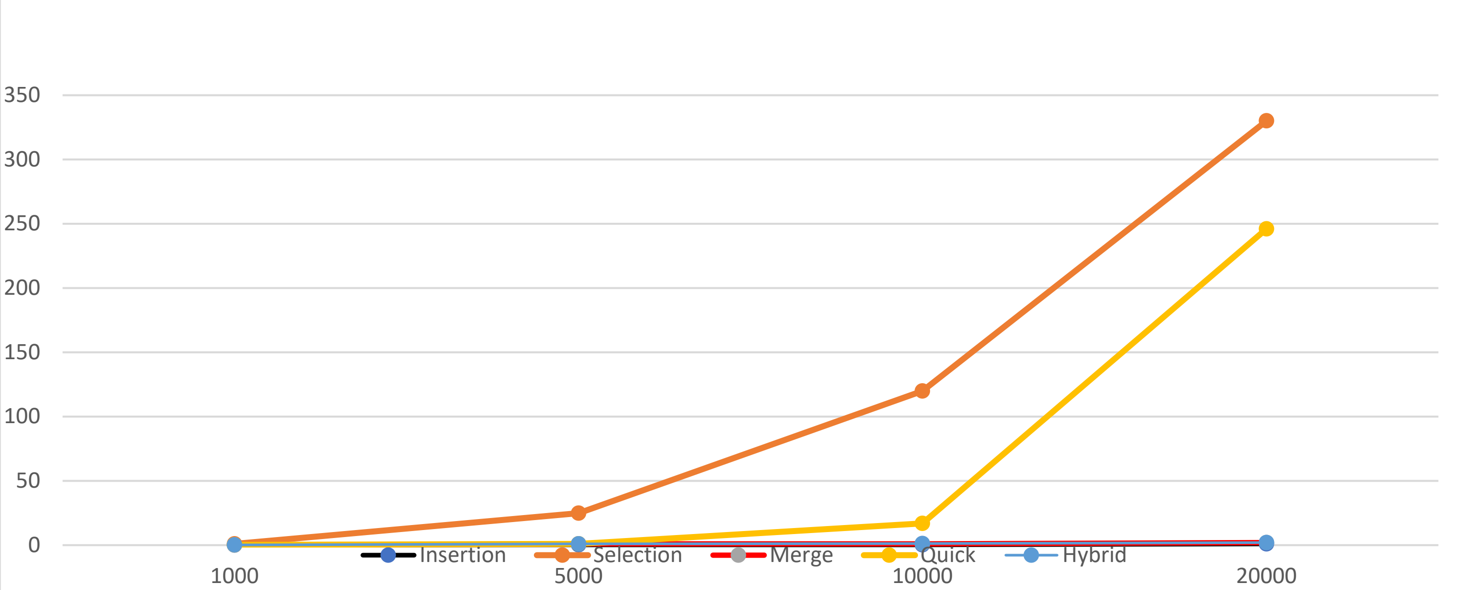
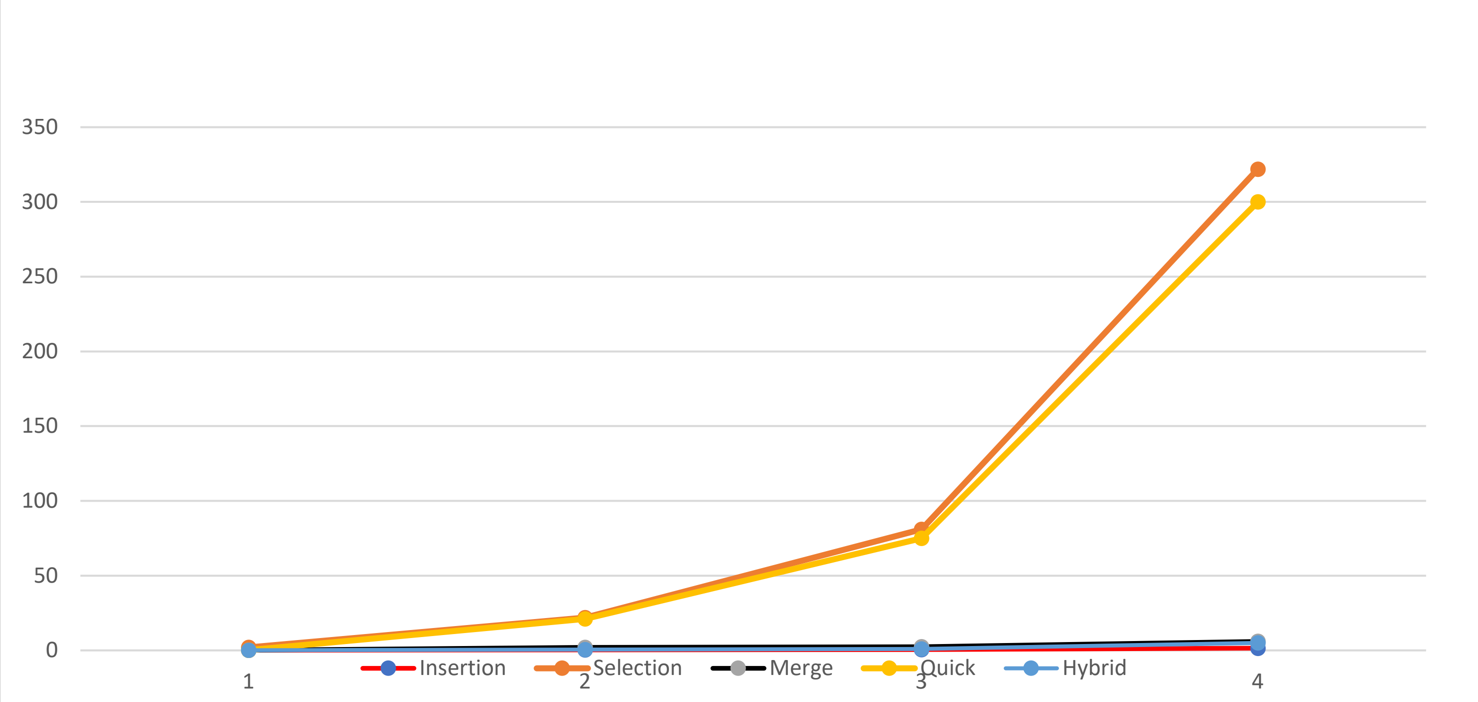
***Title: Algorithm analysis & Sorting Author: Seçkin Alp Kargı ID: 22001942 Section: 1 Homework: 1**

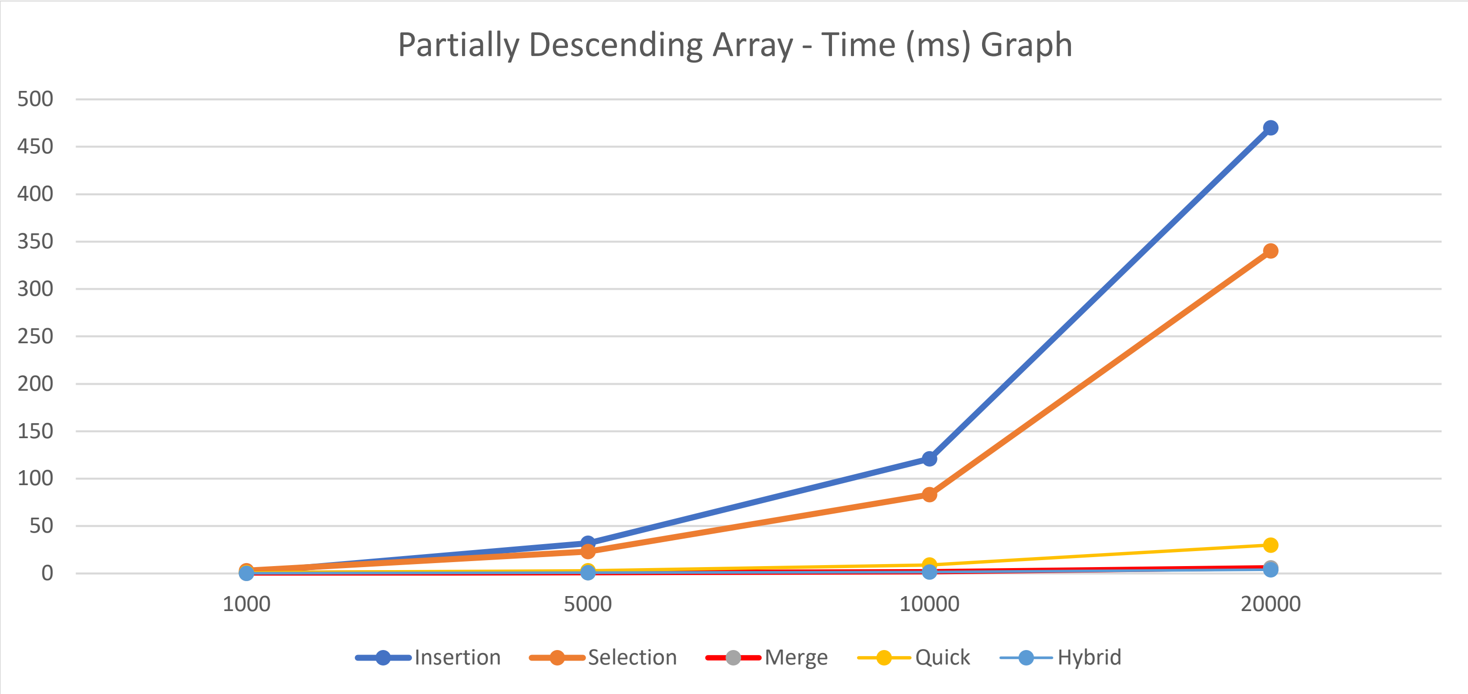
Array	Elapsed Time (ms)					Number of comparisons					Number of Data Moves				
	Insertion	Selection	Merge	Quick	Hybrid	Insertion	Selection	Merge	Quick	Hybrid	Insertion	Selection	Merge	Quick	Hybrid
R1K	0.1	2	0.15	0.2	0.1	3769	499500	8696	10503	8885	1146	2997	19952	18893	20093
R5K	0.4	22	2	21	0.5	29394	12497500	8696	12497500	29782	9998	14997	123616	19996	123440
R10K	0.5	81	2.5	75	1	42221	49995000	64608	49995000	64586	19998	29997	267232	39996	267056
R20K	1.5	322	6	300	5	73231	199990000	139216	199990000	139194	39998	59997	574464	79996	574567
A1K	0.05	1	0.16	0.20	0.15	2237	499500	5804	148094	5940	4235	2997	19952	6623	19880
A5K	0.2	25	1	1	1	22646	12497500	44658	699476	44718	53646	14997	123616	69339	123554
A10K	0.3	120	1.1	17	1.2	39383	49995000	76532	11006403	76589	59381	29997	267232	69838	267170
A20K	1	330	2	246	2	64962	199990000	183866	45934077	183961	104960	59997	574464	137767783	574573
D1K	1	3	0.2	1	0.25	482787	499500	7105	36665	7300	484785	2997	19952	64088	20048
D5K	32	23	1	3	1	12356158	12497500	43420	676730	43510	12366156	14997	123616	1148420	123659
D10K	121	83	2	9	1.5	49669050	49995000	92575	2517495	92679	49689048	29997	267232	4192818	267386
D20K	470	340	6	30	4	199248464	199990000	197489	8372170	197599	199288462	59997	574464	13780450	574546

Partially Ascending Array - Time (ms) Graph



Random Array - Time (ms) Graph





Insertion Sort: Best Case (Array Already Sorted) O(n), Worst Case (Array in reverse) O(n^2).

Selection Sort: Does not depend the array organization. All case O (n^2) key comparisons, O(n) moves.

Merge Sort: Best Case (Elements in the first part are smaller than other part). Key comparison average O(n*log2n)

Bubble Sort: Best Case (Array Already Sorted) O(n), Worst Case (Array in reverse) O(n^2).

Quick Sort: Best Case and Average O(n*log2n). Worst (Array already sorted) O(n^2).

Hybrid Sort: Nearly same as Merge Sort but little bit better.

For small array it is not make a big different, all differences probably because of hardware, processor capabilities, and other system-level factors. It Does the work so fast so in the real life for really small arrays I think it is not important. Selection Sort is worst sorting algorithm because it always gives the Worst time, number of comparison and number of moves. It is because of in all cases case O (n^2) key comparisons, O(n) moves. It does not depend the array Organization so I think nobody should use selection sort if they do not want slow sorting. Insertion Sort has best case (Array Already Sorted) O(n), Worst Case (Array in reverse) O(n^2) so if someone use insertion in best case it is quite fast. Based on the data insertion sort is relatively faster Random and Ascending Arrays I think for random cases using insertion is not reliable but for partially ascending or ascending order arrays insertion is the best choice but it has really big disadvantage in the decreasing order so we must look our arrays orientation carefully when we want to use insertion sort. Quick sort has an average time complexity of O(n log n). It is also a divide-and-conquer algorithm and is widely used due to its efficiency and average-case performance. Quick Sort is good option for best and average case at least it is not high as O(n^2) but in the already sorted array or nearly sorted array orientation it has quite bad performance. Merge Sort also has divide-and-conquer algorithm that performs well for larger input sizes and is generally considered efficient and according to this data we can Clearly see that it is quite well. It is working good in all cases and nearly similar results so it is reliable but hybrid sort is another thing. Hybrid sorting algorithms typically combine the strengths of different sorting algorithms to achieve improved performance and according this data hybrid sort is the winner of this Competition. In the real life if we do not want to modify any sorting algorithm, I think merge sort is quite well and I can prefer merge sort but if we have time And energy for creating our hybrid sorting algorithms Everyone should always choose hybrid sort because it is just focus on the best parts of the sorting algorithms. We can see that bubble sort is better when array size >20 than merge sort but merge sort is better in the other way so we just use this thing and find the best sorting method for this experiment. But if you want to know exact moves and comparison you can use selection sort because it is time and moves is always same and it does not surprise you it has not any random result. If you do not look for speed.

b) [8, 33, 2, 10, 4, 1, 34, 7]

insertion

8 | 33, 2, 10, 4, 1, 34, 7   key = 33, 33>8 done

8, 33 | 2, 10, 4, 1, 34, 7   key = 2, 2<33, shift 2-33, 2<8, shift 2-8 done

2, 8, 33 | 10, 4, 1, 34, 7   key = 10, 10<33, shift 10-33, 10>8 done

2, 8, 10, 33 | 4, 1, 34, 7   key = 4, 4<33, shift 4-33, 4<10, shift 4-10
4<8, shift 4-8, 4>2 done

2, 4, 8, 10, 33 | 1, 34, 7   key = 1, 1<33, shift 1-33, 1<10, shift 1-10, 1<8, shift 1-8
1<4, shift 1-4, 1<2, shift 1-2 done

1, 2, 4, 8, 10, 33 | 34, 7   key = 34, 34>33 done

1, 2, 4, 8, 10, 33, 34 | 7   key = 7, 7<34, shift 7-34, 7<33, shif 7-33
7<10, shift 7-10, 7<8, shift 7-8 7>4 done

1, 2, 4, 7, 10, 33, 34 |

sorted

merge

→ split array into partitions of 1   8-33-2-10-4-1-34-7

merge 8-33   8<33 done → 8, 33-2-10-4-1-34-7

merge 2-10   2<10 done → 8, 33-2, 10-4-1-34-7

merge 8, 33-2, 10   2<8, 8<10, 10<33 done → 2, 8, 10, 33-4-1-34-7

merge 4-1   4>1 shift done → 2, 8, 10, 33-1, 4-34-7

merge 34-7   34>7 shift done → 2, 8, 10, 33-1, 4-7, 34

merge 1, 4-7, 34   1<7, 4<7 done → 2, 8, 10, 33-1, 4, 7, 34

merge 2, 8, 10, 33-1, 4, 7, 34   1<2, 2<4, 4<8, 7<8, 8<34, 10<34, 33<34 done

1, 2, 4, 7, 8, 10, 33, 34

sorted

Quick   chosen 8 as pivot

33>8 next → 8, 33, 2, 10, 4, 1, 34, 7

2<8 swap 2-33 → 8, 2, 33, 10, 4, 1, 34, 7

10>8 next → 8, 2, 33, 10, 4, 1, 34, 7

4<8 swap 4-33 → 8, 2, 4, 10, 33, 1, 34, 7

1<8 swap 1-10 → 8, 2, 4, 1, 33, 10, 34, 7

34>8 next → 8, 2, 4, 1, 33, 10, 34, 7

7<8 swap 7-33 → 8, 2, 4, 1, 7, 10, 34, 33

Swap pilot 7-8 → 7, 2, 4, 1, 8, 10, 34, 33 → pivot sorted ✓

7 is pilot for less than part

7>2 next → 7, 2, 4, 1, 8, 10, 34, 33

7>4 next → 7, 2, 4, 1, 8, 10, 34, 33

7>1 next → 7, 2, 4, 1, 8, 10, 34, 33

swap pilot 7-1 → 1, 2, 4, 7, 8, 10, 34, 33 → 7 sorted ✓ 1 pivot

1<2 next 1<4 next no swap → 1, 2, 4, 7, 8, 10, 34, 33 → 1 sorted ✓ 2 pivot

2<4 next no swap → 1, 2, 4, 7, 8, 10, 34, 33 → 2 sorted ✓ 4 pivot 4 sorted ✓ 10 pivot

10<34 next 10<33 next no swap → 1, 2, 4, 7, 8, 10, 34, 33 → 10 sorted ✓ 34 pivot

34>33 swap 34-33 → 1, 2, 4, 7, 8, 10, 33, 34 → 34 sorted ✓, 33 pivot, 33 sorted ✓

Sorted

a) $f(n) = 8n^4 + 5n^2 - 2n + 4$ $\quad O(n^4)$

- We need to show that there are values for c and n_0 such that $n \geq n_0$ then $8n^4 + 5n^2 - 2n + 4 \leq cn^4$. $n_0 = 2$

$n \geq n_0$, $n \geq 2$, ~~assume~~ $n^4 \geq n^2 \geq n \geq 4$

$8n^4 + 5n^2 - 2n + 4 \leq 8n^4 + 5n^4 - 2n^4 + 4n^4$

$8n^4 + 5n^2 - 2n + 4 \leq 15n^4$

So for $n_0 = 2$ and $c = 15$ we showed that function is complexity is $O(n^2)$

Also we can see that easily if we chose $c \geq 15$ and $n_0 \geq 2$ there are more values.

---

c) $T(n) = T(n/2) + n^2 \quad \cdot T(1) = 1$

$T(n) = T(n/4) + (n/2)^2 + n^2$

$T(n) = T(n/8) + (n/4)^2 + (n/2)^2 + n^2$

$T(n) = n^2 + (n/2)^2 + (n/4)^2 + \dots + (n/2k)^2$

$T(n) = n^2 \left(1 + \frac{1}{4} + \frac{1}{16} + \dots + \frac{1}{2^k} + \frac{1}{2^{k+1}}\right)$

$T(n) = n^2 \underbrace{\sum_{i=0}^{k+1} \frac{1}{2^i}}_{1}$

$T(n) = n^2$

$\underline{O(n^2)}$

$T(n) = T(n/2k) + \frac{n^2}{4^{k-1}} + \frac{n^2}{4^{k-2}} + \dots + \frac{n^2}{4} + n^2$

Assume $\frac{n}{4^k} = 1 \rightarrow n = 4^k$

$k = \log n$

$T(n) = T(1) + n^2\left(\frac{1}{4^{k-1}} + \frac{1}{4^{k-2}} + \dots \frac{1}{4} + 1\right)$

$T(n) = 1 + n^2(1 + 1)$

$T(n) = 1 + 2n^2 \longrightarrow \underline{O(n^2)}$