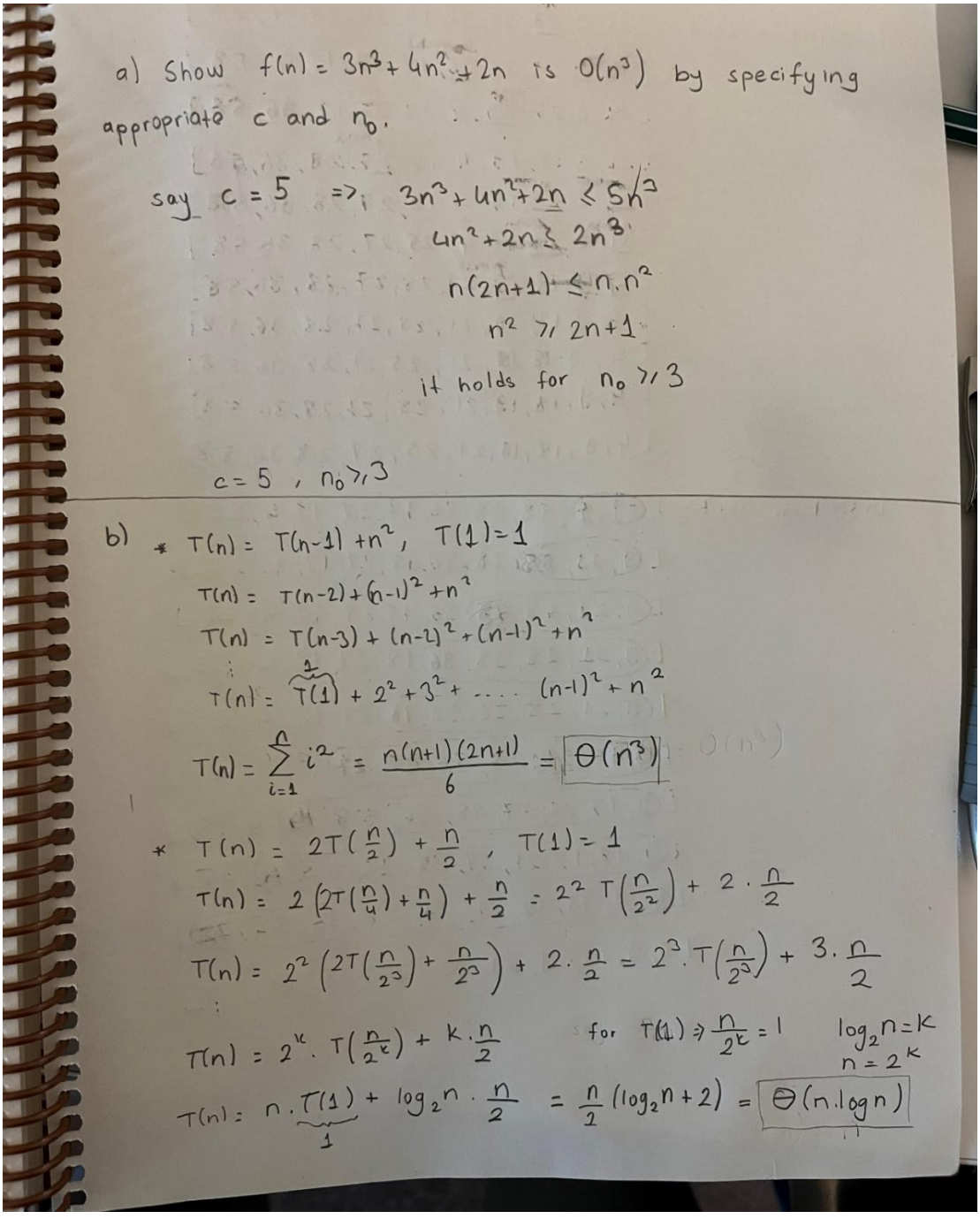
CS202-1, Fall 2022

Homework 1

İdil Atmaca 22002491

Question 1



a) Show $f(n) = 3n^3 + 4n^2 + 2n$ is $O(n^3)$ by specifying appropriate c and n_0.

say $c = 5$ $\Rightarrow$ $3n^3 + 4n^2 + 2n \leq 5n^3$

$4n^2 + 2n \geq 2n^3$

$n(2n+1) \leq n \cdot n^2$

$n^2 \geq 2n+1$

it holds for $n_0 \geq 3$

$c = 5$, $n_0 \geq 3$

b) * $T(n) = T(n-1) + n^2$, $T(1) = 1$

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

$T(n) = T(n-3) + (n-2)^2 + (n-1)^2 + n^2$

$T(n) = \overbrace{T(1)}^{1} + 2^2 + 3^2 + \dots + (n-1)^2 + n^2$

$T(n) = \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \boxed{\Theta(n^3)}$

* $T(n) = 2T(\frac{n}{2}) + \frac{n}{2}$, $T(1) = 1$

$T(n) = 2\left(2T(\frac{n}{4}) + \frac{n}{4}\right) + \frac{n}{2} = 2^2 T\left(\frac{n}{2^2}\right) + 2 \cdot \frac{n}{2}$

$T(n) = 2^2\left(2T(\frac{n}{2^3}) + \frac{n}{2^3}\right) + 2 \cdot \frac{n}{2} = 2^3 \cdot T\left(\frac{n}{2^3}\right) + 3 \cdot \frac{n}{2}$

$T(n) = 2^k \cdot T\left(\frac{n}{2^k}\right) + k \cdot \frac{n}{2}$ for $T(1) \Rightarrow \frac{n}{2^k} = 1$ $\log_2 n = k$, $n = 2^k$

$T(n) = n \cdot \underbrace{T(1)}_{1} + \log_2 n \cdot \frac{n}{2} = \frac{n}{2}(\log_2 n + 2) = \boxed{\Theta(n \log n)}$

c) Selection Sort: [21, 9, 58, 28, 36, 18, 27, 19, 4, 25]

[21, 9, 25, 28, 36, 18, 27, 19, 4, 58]

[21, 9, 25, 28, 4, 18, 27, 19, 36, 58]

[21, 9, 25, 19, 4, 18, 27, 28, 36, 58]

[21, 9, 25, 19, 4, 18, 27, 28, 36, 58]

[21, 9, 18, 19, 4, 25, 27, 28, 36, 58]

[4, 9, 18, 19, 21, 25, 27, 28, 36, 58]

[4, 9, 18, 19, 21, 25, 27, 28, 36, 58]

[4, 9, 18, 19, 21, 25, 27, 28, 36, 58]

[4, 9, 18, 19, 21, 25, 27, 28, 36, 58]

Insertion Sort: [21, 9, 58, 28, 36, 18, 27, 19, 4, 25]

[9, 21, 58, 28, 36, 18, 27, 19, 4, 25]

[9, 21, 58, 28, 36, 18, 27, 19, 4, 25]

[9, 21, 28, 58, 36, 18, 27, 19, 4, 25]

[9, 21, 28, 36, 58, 18, 27, 19, 4, 25]

[9, 18, 21, 28, 36, 58, 27, 19, 4, 25]

[9, 18, 21, 27, 28, 36, 58, 19, 4, 25]

[9, 18, 19, 21, 27, 28, 36, 58, 4, 25]

[4, 9, 18, 19, 21, 27, 28, 36, 58, 25]

[4, 9, 18, 19, 21, 25, 27, 28, 36, 58]

Question 2- outputs

```
Testing for the  ARRAY: 40 25 29 56 37 27 24 32 79 12 35 38 23 31 33 26
Testing bubble sort algorithm
Sorted ARRAY: 12 23 24 25 26 27 29 31 32 33 35 37 38 40 56 79
comparison count: 114
move count: 204

Testing merge sort algorithm
Sorted ARRAY: 12 23 24 25 26 27 29 31 32 33 35 37 38 40 56 79
comparison count: 46
move count: 128

Testing quick sort algorithm
Sorted ARRAY: 12 23 24 25 26 27 29 31 32 33 35 37 38 40 56 79
comparison count: 48
move count: 105
```

```
PERFORMANCE ANALYSIS FOR RANDOMLY GENERATED ARRAYS
---------------------------------------
Algorithm Analysis for Bubble Sort
Array Size    Elapsed time    compCount    moveCount
4000          92.4972         7995225        12048096
8000          415.136         31992514       47974338
12000          975.458         71979635       109502193
16000          1773.81         127973855       191839524
20000          2815.34         199980130       301834467
24000          4081.77         287916747       431389797
28000          5624.56         391899680       589179015
32000          7354.23         511848019       769467843
36000          9449.98         647977344       967671630
40000          11622.7         799958264       1193508219
44000          14154.4         967951665       1450571304
48000          17031.9         1151972345       1720010772
---------------------------------------
Algorithm Analysis for Merge Sort
Array Size    Elapsed time    compCount    moveCount
4000          1.14966         42875        95808
8000          2.44093         93584        207616
12000          3.82672         147579        327232
16000          5.21162         203384        447232
20000          6.70705         260846        574464
24000          8.1335         319408        702464
28000          9.62147         378659        830464
32000          11.3559         438501        958464
36000          13.0671         500096        1092928
40000          14.7351         561955        1228928
44000          15.7494         624036        1364928
48000          17.5951         686870        1500928
---------------------------------------
Algorithm Analysis for Quick Sort
Array Size    Elapsed time    compCount    moveCount
4000          0.933537         54070        84978
8000          2.00676         127365        184176
12000          3.25478         200702        331689
16000          4.28599         256978        384666
20000          5.6151         341802        554208
24000          6.83139         434622        646665
28000          8.00295         484386        732168
32000          9.52479         578225        1027665
36000          10.4469         632732        998523
40000          11.7825         696099        1025376
44000          13.1207         820081        1198827
48000          15.3639         936977        1585410
```

PERFORMANCE ANALYSIS FOR RANDOMLY GENERATED DESCENDING ARRAYS

Algorithm Analysis for Bubble Sort

Array Size	Elapsed time	compCount	moveCount
4000	84.8436	7992747	12060753
8000	344.278	31993585	48165168
12000	781.501	71991855	108288381
16000	1415.28	127977972	192345744
20000	2180.27	199967845	300050127
24000	3135.52	287977122	432403263
28000	4280.61	391971804	586927200
32000	5582.82	511967529	767876574
36000	7095.76	647977814	971616564
40000	8765.44	799975247	1204659063
44000	10567.2	967964139	1455818838
48000	12662.5	1151936097	1730460639

Algorithm Analysis for Merge Sort

Array Size	Elapsed time	compCount	moveCount
4000	0.924646	40104	95808
8000	1.99948	88486	207616
12000	3.38412	138511	327232
16000	4.64107	193593	447232
20000	5.53917	245798	574464
24000	6.8702	302127	702464
28000	8.72804	359523	830464
32000	9.43707	419745	958464
36000	10.7346	475694	1092928
40000	12.7277	532830	1228928
44000	14.3644	591802	1364928
48000	15.8213	653829	1500928

Algorithm Analysis for Quick Sort

Array Size	Elapsed time	compCount	moveCount
4000	0.897255	56582	87066
8000	1.98386	132717	195153
12000	3.11193	195450	311784
16000	4.39525	295027	421929
20000	5.60792	365811	577851
24000	6.62088	429345	686397
28000	7.99287	504151	805266
32000	9.27406	616538	970617
36000	10.1994	663797	1020633
40000	11.8596	765862	1198089
44000	13.0629	832128	1353126
48000	14.1892	916822	1373673

PERFORMANCE ANALYSIS FOR RANDOMLY GENERATED ASCENDING ARRAYS

Algorithm Analysis for Bubble Sort

Array Size	Elapsed time	compCount	moveCount
4000	0.02072	3999	0
8000	0.041461	7999	0
12000	0.06216	11999	0
16000	0.082565	15999	0
20000	0.103311	19999	0
24000	0.123925	23999	0
28000	0.144532	27999	0
32000	0.165281	31999	0
36000	0.18981	35999	0
40000	0.206891	39999	0
44000	0.230937	43999	0
48000	0.247827	47999	0

Algorithm Analysis for Merge Sort

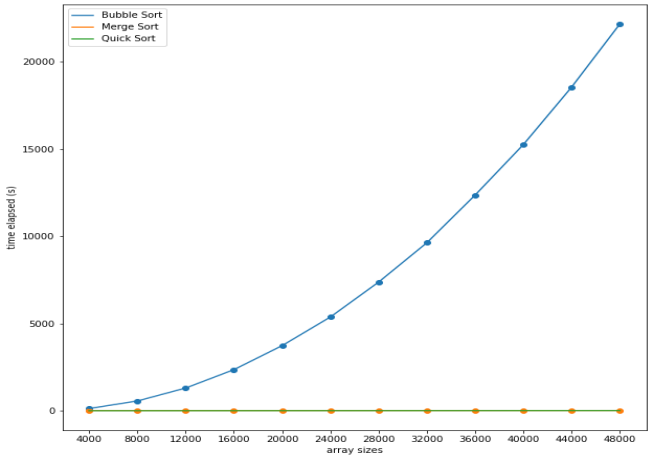
Array Size	Elapsed time	compCount	moveCount
4000	0.754295	24177	95808
8000	1.50989	52353	207616
12000	2.36109	84304	327232
16000	3.17893	112704	447232
20000	4.10414	148016	574464
24000	4.98887	180608	702464
28000	5.86531	212720	830464
32000	6.72098	241408	958464
36000	7.68167	279184	1092928
40000	8.67544	316032	1228928
44000	9.60525	352048	1364928
48000	10.5259	385216	1500928

Algorithm Analysis for Quick Sort

Array Size	Elapsed time	compCount	moveCount
4000	35.1419	7998000	11997
8000	135.813	31996000	23997
12000	305.586	71994000	35997
16000	543.403	127992000	47997
20000	848.845	199990000	59997
24000	1222.56	287988000	71997
28000	1664.23	391986000	83997
32000	2173.62	511984000	95997
36000	2750.73	647982000	107997
40000	3395.74	799980000	119997
44000	4107.83	967978000	131997
48000	4888.69	1151976000	143997

Question 3

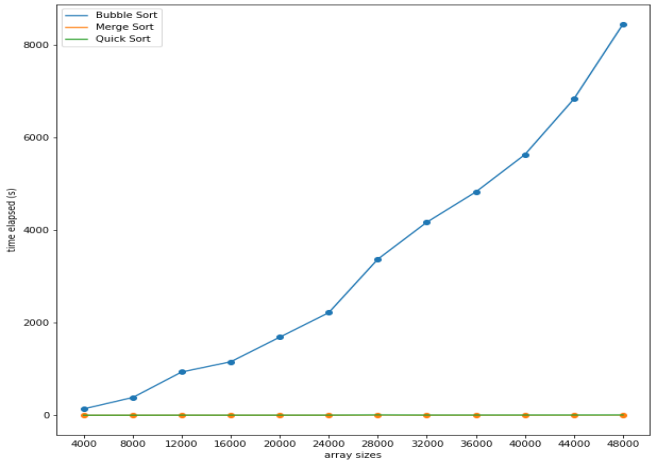
Algorithm Analysis graph for randomly generated arrays



For ***Bubble Sort***, as it was expected, time complexity was $\mathbf{O(n^2)}$ since it is the average case for this algorithm.

As for ***Merge Sort*** and ***Quick Sort***, time complexity was approximately $\mathbf{O(n*\log_2 n)}$. Since their time complexity was almost the same for these data, we see an overlap on their lines.

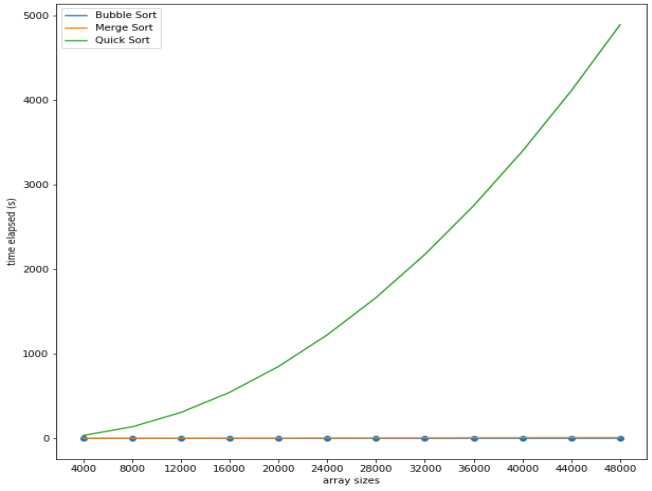
Algorithm Analysis graph for randomly descending generated arrays



Again for ***Bubble Sort*** we see a time complexity of $\mathbf{O(n^2)}$ which is the worst case for this algorithm since it is in descending order.

We do not see a significant difference between descending and random array cases for ***Merge Sort*** and ***Quick Sort***, it is $\mathbf{O(n*\log_2 n)}$.

Algorithm Analysis graph for randomly ascending generated arrays



For ***Bubble Sort*** we encounter best case for already sorted arrays. Time complexity is $\mathbf{O(n)}$.

As for ***Merge Sort*** time complexity is again $\mathbf{O(n*\log_2 n)}$. It looks like these two graphs overlap but it is due to how slow ***Quick Sort*** is.

For ***Quick Sort***, it is the worst case and the time complexity was approximately $\mathbf{O(n^2)}$.