a) (15 points) Write down the asymptotic upper bound for the Quicksort for best case, worst case and average case. Prove them solving the recurrence equations.

Worst case running time: $\Theta(n^2)$

When the array/list is already sorted either ascending or descending order the worst case happens.

Prove of it: Total of the partitioning times for each level is equal to

T(n) = n + T(n-1)

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

T(n-2) = n-2 + T(n-3)

….

T(2) = 2 + T(1)

T(1) = 0

When all the equations are sum

T(n) = n + n-1 + n-2 + … + 2 + 0

T(n) = n + n-1 + n-2 + … + 2 + 1 – 1

T(n) = (n.(n+1) / 2) – 1

T(n) = n^2/2 + n/2 – 1

T(n) = n^2

Biggest term is n^2due to that big-O notation is O(n^2)

Average case running time: O(n.log(n))

When the pivot element splits the list into two equal halves which may be different sizes the average case happens.

Prove of it:

T(n) ≤ T(i) + T(n-i-1) + c.n

While divide-conquer tree drawn, one of the left-right child will be have more depth cause we are checking average value, if i == n – i ,t will be best case

If i > n – i, the depth of the left side will be bigger than right side. Since each child is i/n of the size of the node above(parent), each parent is n/i times the size of the left child. That means, from starting the leaf(size == 1) we are reaching n with multiplying n/i k times(depth).

1.$(n/i)^k$ = n ⟶ k = $\log_{n/i} n$

Each time we divide n/i, we will get c.n because each level has c.n work to do. And also owing to i is some part of n, we can write

i = n/a (a is a fractional number)

While combining this informations, total work to do is k.c.n

k.c.n = $\log_{n/i}(n)$.c.n ⟶ $\log_a(n)$.c.n ⟶ n.log(n)

i can take every value that n -1 can take. So this equation will be true also for n – i > i

So total partitioning time big-O notation is O(n.log(n))

Best case running time: O(n.log(n))

When the pivot element splits the list into two equal halves by being exactly in the middle position the best case happens.

Prove of it: Tip: Problem size is equal to the total of subproblem sizes which are equal

k times

$T(n) = 2.T(n/2) + n$

$T(n/2) = 2.T(n/4) + n/2 \longrightarrow T(n) = 2[\ 2.T(n/4) + n/2] + n$

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

$T(n/4) = 2.T(n/8) + n/4 \longrightarrow T(n) = 2[\ 2.[\ 2.T(n/8) + n/4] + n/2] + n$

$= 2^3.T(n/2^3) + 3.n$

…

$T(2) = 2.T(1) + n/2^k. \longrightarrow T(n) = 2^k.T(1) + k.n$

$T(1) = 0 \longrightarrow T(n) = k.n$

Each time we divide n→n/2 so 2^k=n $\longrightarrow \log_2 n$

So time complexity $T(n) = n.\log_2(n)$

So total partitioning time big-O notation is O(n.log(n))

b)

1) Assume that at first we have a input case as below

sorted (5).txt - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

Country	Item Type	Order ID	Units Sold	Total Profit
Turkey	Baby Food	707877290	5475	524834
Afghanistan	Snacks	467044961	7838	432187
Canada	Cosmetics	140492665	3757	653230
Iceland	Personal Care	179688537	2686	67311.2
Afghanistan	Fruits	787060821	7642	18417.2
Zimbabwe	Fruits	886772906	4664	11240.2
Bangladesh	Beverages	659767472	8711	136414
Afghanistan	Vegetables	837767016	2732	172471.16
Bangladesh	Clothes	388088881	2979	218777.76
Bangladesh	Office Supplies	743760097	3995	504368.75

At first we will sort it by profits and save this to sorted_by_profits.txt

sorted_by_profits (1).txt - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

Country	Item Type	Order ID	Units Sold	Total Profit
Canada	Cosmetics	140492665	3757	653230
Turkey	Baby Food	707877290	5475	524834
Bangladesh	Office Supplies	743760097	3995	504369
Afghanistan	Snacks	467044961	7838	432187
Bangladesh	Clothes	388088881	2979	218778
Afghanistan	Vegetables	837767016	2732	172471
Bangladesh	Beverages	659767472	8711	136414
Iceland	Personal Care	179688537	2686	67311.2
Afghanistan	Fruits	787060821	7642	18417.2
Zimbabwe	Fruits	886772906	4664	11240.2

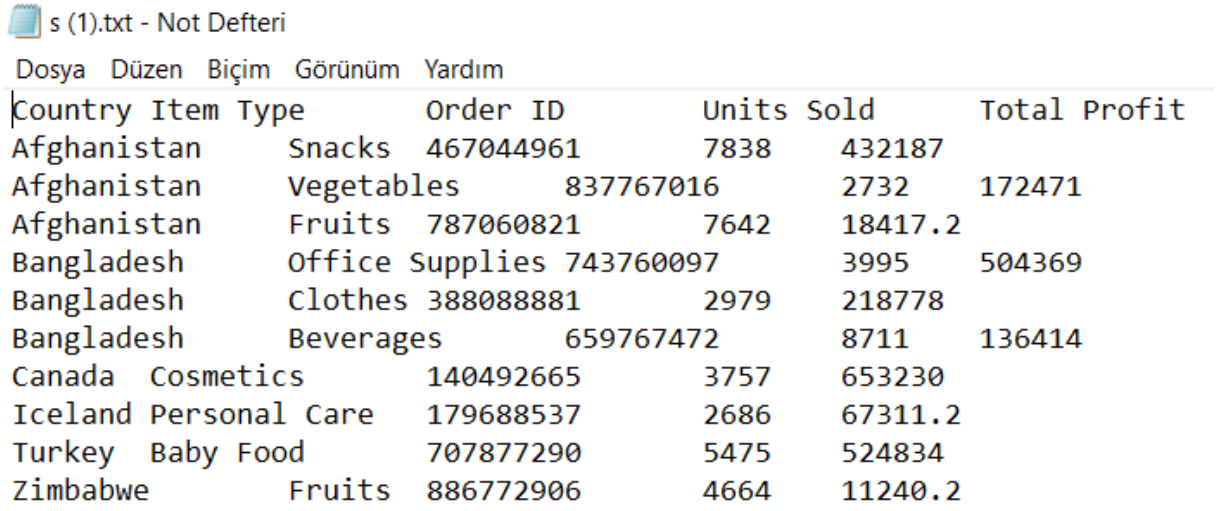
Then we will sort this input case by ascending order of country name

sorted_end.txt - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

Country	Item Type	Order ID	Units Sold	Total Profit
Afghanistan	Fruits	787060821	7642	18417.2
Afghanistan	Vegetables	837767016	2732	172471
Afghanistan	Snacks	467044961	7838	432187
Bangladesh	Beverages	659767472	8711	136414
Bangladesh	Clothes	388088881	2979	218778
Bangladesh	Office Supplies	743760097	3995	504369
Canada	Cosmetics	140492665	3757	653230
Iceland	Personal Care	179688537	2686	67311.2
Turkey	Baby Food	707877290	5475	524834
Zimbabwe	Fruits	886772906	4664	11240.2

As it can be seen, output case is sorted by names but not profits. The output must be like:


s (1).txt - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

Country	Item Type	Order ID	Units Sold	Total Profit
Afghanistan	Snacks	467044961	7838	432187
Afghanistan	Vegetables	837767016	2732	172471
Afghanistan	Fruits	787060821	7642	18417.2
Bangladesh	Office Supplies	743760097	3995	504369
Bangladesh	Clothes	388088881	2979	218778
Bangladesh	Beverages	659767472	8711	136414
Canada	Cosmetics	140492665	3757	653230
Iceland	Personal Care	179688537	2686	67311.2
Turkey	Baby Food	707877290	5475	524834
Zimbabwe	Fruits	886772906	4664	11240.2

This implementation showed that given solution does **NOT** gives the desired output.

2)Insertion sort, merge sort, selection sort. Those three algorithms will give the desired output.

c)

Average working time while reading sales.txt (second):

```
[atlamaz18@ssh 335-1]$ ./a.out 10
0.000077 sec
[atlamaz18@ssh 335-1]$ ./a.out 10
0.000079 sec
[atlamaz18@ssh 335-1]$ ./a.out 10
0.000080 sec
[atlamaz18@ssh 335-1]$ ./a.out 10
0.000125 sec
[atlamaz18@ssh 335-1]$ ./a.out 10
0.000083 sec
[atlamaz18@ssh 335-1]$ ./a.out 10
0.000056 sec
[atlamaz18@ssh 335-1]$ ./a.out 10
0.000122 sec
[atlamaz18@ssh 335-1]$ ./a.out 10
0.000082 sec
[atlamaz18@ssh 335-1]$ ./a.out 10
0.000137 sec
[atlamaz18@ssh 335-1]$ ./a.out 10
0.000078 sec
```

```
[atlamaz18@ssh 335-1]$ ./a.out 100
0.000113 sec
[atlamaz18@ssh 335-1]$ ./a.out 100
0.000183 sec
[atlamaz18@ssh 335-1]$ ./a.out 100
0.000103 sec
[atlamaz18@ssh 335-1]$ ./a.out 100
0.000113 sec
[atlamaz18@ssh 335-1]$ ./a.out 100
0.000140 sec
[atlamaz18@ssh 335-1]$ ./a.out 100
0.000100 sec
[atlamaz18@ssh 335-1]$ ./a.out 100
0.000103 sec
[atlamaz18@ssh 335-1]$ ./a.out 100
0.000113 sec
[atlamaz18@ssh 335-1]$ ./a.out 100
0.000258 sec
[atlamaz18@ssh 335-1]$ ./a.out 100
0.000108 sec
```

```
[atlamaz18@ssh 335-1]$ ./a.out 1000
0.000791 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000
0.000996 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000
0.000809 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000
0.000758 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000
0.000796 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000
0.000767 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000
0.000956 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000
0.000903 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000
0.000776 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000
0.000750 sec
```

```
[atlamaz18@ssh 335-1]$ ./a.out 100000
0.195404 sec
[atlamaz18@ssh 335-1]$ ./a.out 100000
0.231962 sec
[atlamaz18@ssh 335-1]$ ./a.out 100000
0.203620 sec
[atlamaz18@ssh 335-1]$ ./a.out 100000
0.257245 sec
[atlamaz18@ssh 335-1]$ ./a.out 100000
0.429001 sec
[atlamaz18@ssh 335-1]$ ./a.out 100000
0.223026 sec
[atlamaz18@ssh 335-1]$ ./a.out 100000
0.281835 sec
[atlamaz18@ssh 335-1]$ ./a.out 100000
0.358823 sec
[atlamaz18@ssh 335-1]$ ./a.out 100000
0.312540 sec
[atlamaz18@ssh 335-1]$ ./a.out 100000
0.299126 sec
```

```
[atlamaz18@ssh 335-1]$ ./a.out 500000
1.492305 sec
[atlamaz18@ssh 335-1]$ ./a.out 500000
1.492301 sec
[atlamaz18@ssh 335-1]$ ./a.out 500000
1.443767 sec
[atlamaz18@ssh 335-1]$ ./a.out 500000
1.806216 sec
[atlamaz18@ssh 335-1]$ ./a.out 500000
1.682783 sec
[atlamaz18@ssh 335-1]$ ./a.out 500000
1.646200 sec
[atlamaz18@ssh 335-1]$ ./a.out 500000
2.862448 sec
[atlamaz18@ssh 335-1]$ ./a.out 500000
1.583737 sec
[atlamaz18@ssh 335-1]$ ./a.out 500000
1.728224 sec
[atlamaz18@ssh 335-1]$ ./a.out 500000
1.662474 sec
```

```
[atlamaz18@ssh 335-1]$ ./a.out 1000000
3.077892 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000000
3.507167 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000000
4.462722 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000000
3.908109 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000000
3.600829 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000000
4.360191 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000000
3.961157 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000000
4.140199 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000000
4.369454 sec
[atlamaz18@ssh 335-1]$ ./a.out 1000000
4.204149 sec
```

N = 10 -> Average time is 0.000092 second.

N = 100 -> Average time is 0.000133 second.

N = 1000 -> Average time is 0.000830 second.

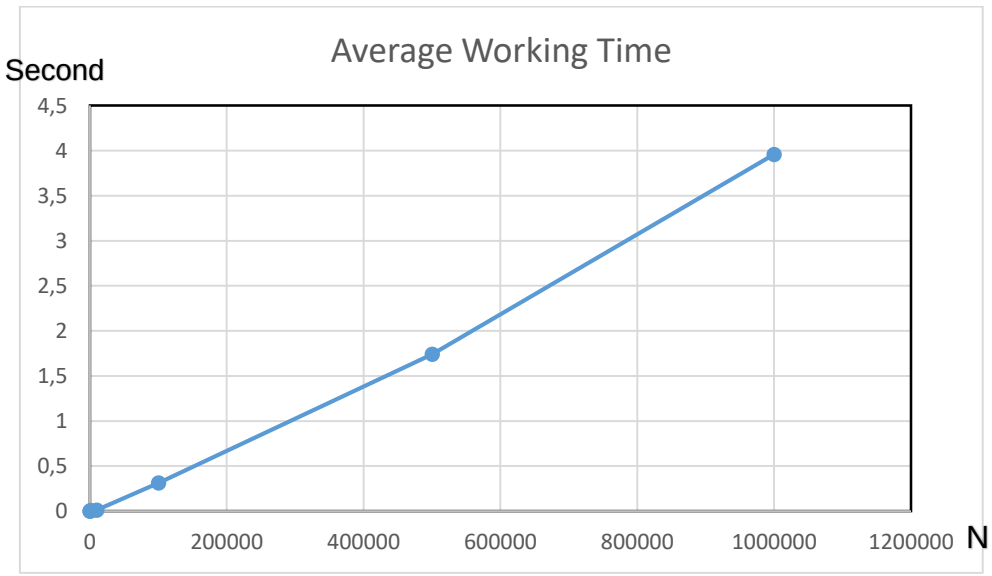
N = 10K -> Average time is 0.012147 second.

N = 100K -> Average time is 0.311861 second.

N = 500K -> Average time is 1.740046 second.

N = 1M -> Average time is 3.959187 second.

	Average Working Time
N = 10	0.000092 second
N = 100	0.000133 second
N = 1000	0.000830 second
N = 10K	0.012147 second
N = 100K	0.311861 second
N = 500K	1.740046 second.
N = 1M	3.959187 second.



Reading sales.txt is the average case due to all of the elements are seperated randomly before we sort them. Time complexicity is n.log(n) and when we compare the results we actually can see it. For example, if N goes from 1000 to 10k, measured times increasing 14,63 times higher, else if N goes from 10k to 100k, measured times increasing 25,67 times higher. Increasing multiplier increses due to T(n) > c.n, but still it is smaller than T(n) < n^2 because it is not the worst case. So time complexicity must be n.log(n)

d)

Average working time while reading sorted.txt (second):

```
[atlamazl8@ssh 335-1]$ ./a.out 10
0.000094 sec
[atlamazl8@ssh 335-1]$ ./a.out 10
0.000073 sec
[atlamazl8@ssh 335-1]$ ./a.out 10
0.000073 sec
[atlamazl8@ssh 335-1]$ ./a.out 10
0.000077 sec
[atlamazl8@ssh 335-1]$ ./a.out 10
0.000071 sec
[atlamazl8@ssh 335-1]$ ./a.out 10
0.000101 sec
[atlamazl8@ssh 335-1]$ ./a.out 10
0.000069 sec
[atlamazl8@ssh 335-1]$ ./a.out 10
0.000121 sec
[atlamazl8@ssh 335-1]$ ./a.out 10
0.000075 sec
[atlamazl8@ssh 335-1]$ ./a.out 10
0.000130 sec
```

```
[atlamazl8@ssh 335-1]$ ./a.out 100
0.000382 sec
[atlamazl8@ssh 335-1]$ ./a.out 100
0.000342 sec
[atlamazl8@ssh 335-1]$ ./a.out 100
0.000442 sec
[atlamazl8@ssh 335-1]$ ./a.out 100
0.000349 sec
[atlamazl8@ssh 335-1]$ ./a.out 100
0.000298 sec
[atlamazl8@ssh 335-1]$ ./a.out 100
0.000267 sec
[atlamazl8@ssh 335-1]$ ./a.out 100
0.000380 sec
[atlamazl8@ssh 335-1]$ ./a.out 100
0.000290 sec
[atlamazl8@ssh 335-1]$ ./a.out 100
0.000377 sec
[atlamazl8@ssh 335-1]$ ./a.out 100
0.000257 sec
```

```
[atlamazl8@ssh 335-1]$ ./a.out 1000
0.054330 sec
[atlamazl8@ssh 335-1]$ ./a.out 1000
0.078205 sec
[atlamazl8@ssh 335-1]$ ./a.out 1000
0.043020 sec
[atlamazl8@ssh 335-1]$ ./a.out 1000
0.065756 sec
[atlamazl8@ssh 335-1]$ ./a.out 1000
0.020965 sec
[atlamazl8@ssh 335-1]$ ./a.out 1000
0.126144 sec
[atlamazl8@ssh 335-1]$ ./a.out 1000
0.072911 sec
[atlamazl8@ssh 335-1]$ ./a.out 1000
0.103464 sec
[atlamazl8@ssh 335-1]$ ./a.out 1000
0.076537 sec
[atlamazl8@ssh 335-1]$ ./a.out 1000
0.075574 sec
```

```
[atlamazl8@ssh 335-1]$ ./a.out 10000
13.835370 sec
[atlamazl8@ssh 335-1]$ ./a.out 10000
14.429267 sec
[atlamazl8@ssh 335-1]$ ./a.out 10000
13.475059 sec
[atlamazl8@ssh 335-1]$ ./a.out 10000
10.975936 sec
[atlamazl8@ssh 335-1]$ ./a.out 10000
11.309948 sec
[atlamazl8@ssh 335-1]$ ./a.out 10000
10.886086 sec
[atlamazl8@ssh 335-1]$ ./a.out 10000
16.218607 sec
[atlamazl8@ssh 335-1]$ ./a.out 10000
9.778258 sec
[atlamazl8@ssh 335-1]$ ./a.out 10000
11.027958 sec
[atlamazl8@ssh 335-1]$ ./a.out 10000
7.714157 sec
```

	Average Working Time
N = 10	0.000088 second
N = 100	0.000304 second
N = 1000	0.071691 second
N = 10K	11.965065 second.

N = 10 -> Average time is 0.000088 second.

N = 100 -> Average time is 0.000304 second.

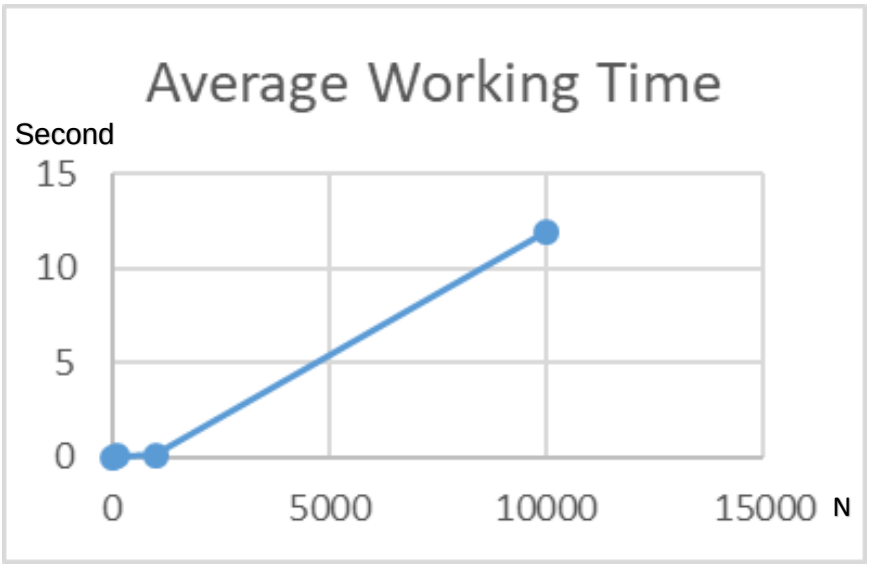
N = 1000 -> Average time is 0.071691 second.

N = 10K -> Average time is 11.965065 second.

N = 100K -> KILLED

N = 500K -> KILLED

N = 1M -> KILLED



a) Reading sorted.txt is the worst case due to all of the elements are sorted already. So while sorting the elements that are already sorted, the divide-conquer method will divide the list two parts,1-(n-1) and this situation will maximize the total time. In total we are dividing the list n – 1 times and for example if we divide 3 times, in the third time we are checking n – 3 elements. This situation maximizes total cost and time and if the data set has large elements the compiler gives KILLED error.

b) While the list is sorted, total time is maximizing and becomes to the worst case; so as much the input case is sorted we will get more similar results.

c) If the input case is sorted and it is known, we can use different algorithms, or we can select the pivot randomly, not from head or tail.

Burak Atlamaz

150180067