Assignment 2

Chitrarth Singh
A20387080

1.

Chitrarth Singh

Assignment-2

1(a) Given : $\pi = (\pi_1, \pi_2, \pi_3 \ldots \pi_n)$ be a random permutation of $\{1, 2 \ldots n\}$

To find:- The expected value of

$$\frac{1}{n} \sum_{i=1}^{n} |\pi_i - i|$$

Averaging around i, we get:

$$\Rightarrow \frac{1}{n} \sum_{\pi_i=1}^{n} |\pi_i - i|$$

$$= \frac{1}{n} \left[\sum_{\pi_i=1}^{i} (i - \pi_i) + \sum_{\pi_i=i+1}^{n} (\pi_i - i) \right]$$

$$= \frac{1}{n} \left[\sum_{\pi_i=1}^{i-1} \pi_i + \sum_{\pi_i=1}^{n-i} \pi_i \right] \text{ (split the intervals)}$$

$$= \frac{1}{n} \left[\frac{(i-1)(i-1+1)}{2} + \frac{(n-i)(n-i+1)}{2} \right]$$

$$= \frac{1}{2n} \left(i^2 - i + n^2 - ni + n - in + i^2 - i \right)$$

$$= \frac{1}{2n} \left(2i^2 - 2i + n^2 - 2in + n \right)$$

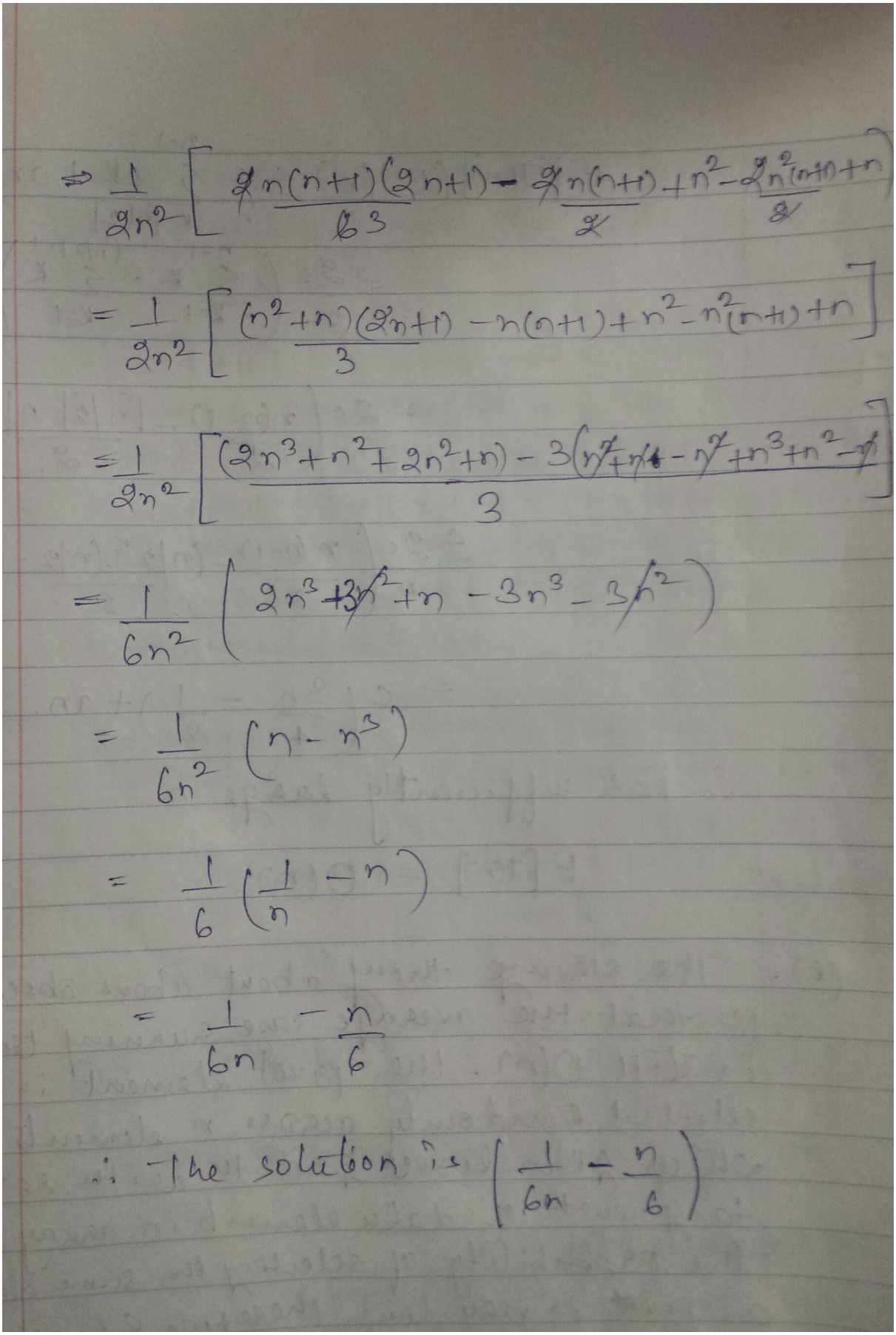
Averaging over i:-

$$= \frac{1}{2n^2} \sum_{i=1}^{n} \left(2i^2 - 2i + n^2 - 2in + n \right)$$

Assignment 2

Chitrarth Singh
A20387080

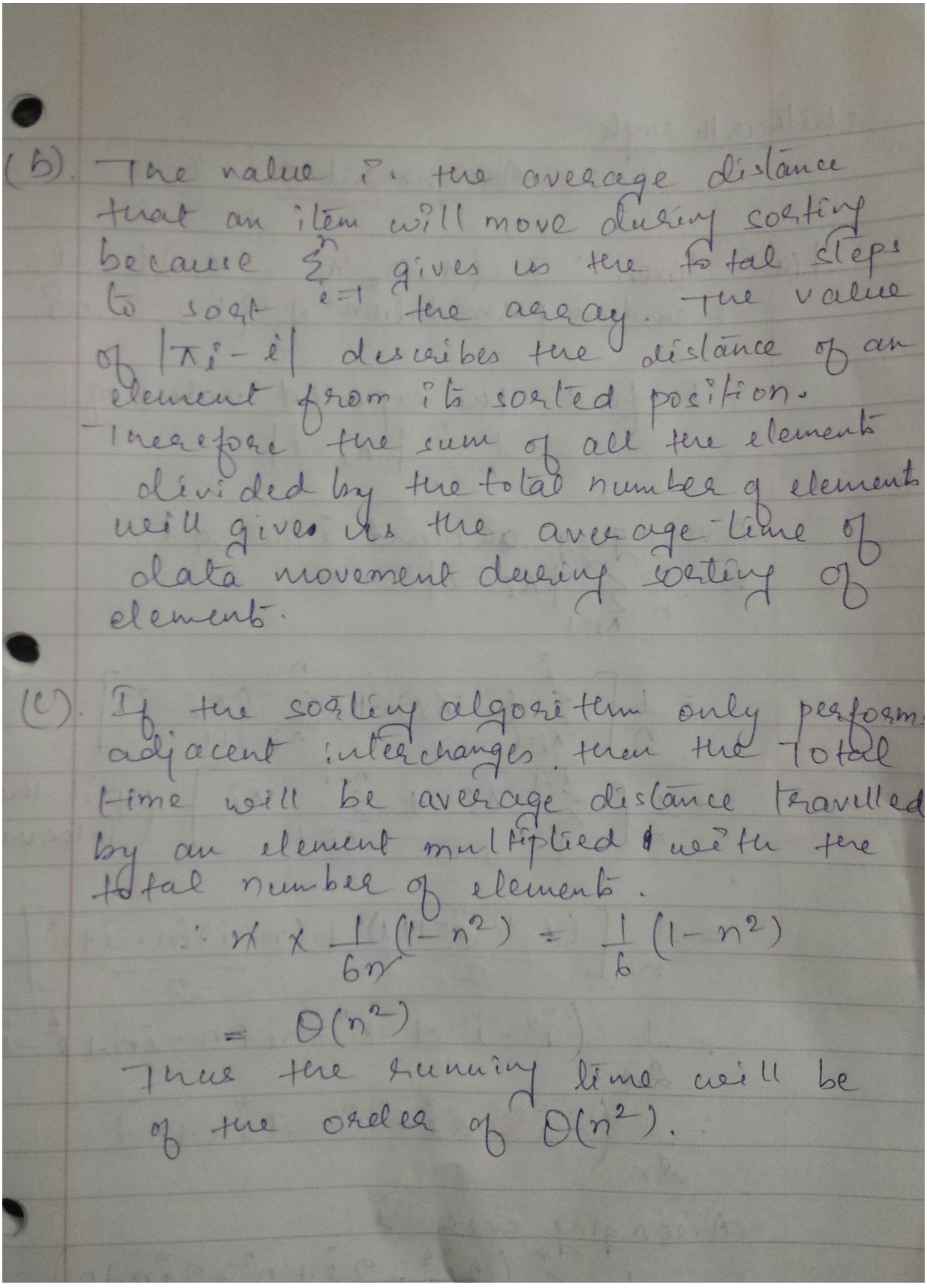
1 continue:



Assignment 2

Chitrarth Singh
A20387080

1 continue:



(b). The value i is the average distance that an item will move during sorting because $\sum_{i=1}^{n}$ gives us the total steps to sort the array. The value of $|\pi_i - i|$ describes the distance of an element from its sorted position. Therefore the sum of all the elements divided by the total number of elements will gives us the average time of data movement during sorting of elements.

(c). If the sorting algorithm only performs adjacent interchanges then the total time will be average distance travelled by an element multiplied with the total number of elements.

$$\therefore n \times \frac{1}{6n}(1-n^2) = \frac{1}{6}(1-n^2)$$

$$= \Theta(n^2)$$

Thus the running time will be of the order of $\Theta(n^2)$.

Assignment 2

Chitrarth Singh
A20387080

2.

2. When the array A of length is already sorted in decreasing order, the largest element is at the root which makes it the best case scenario. The Build-Max-Heap procedure will still run through to ensure max-Heap property is still satisfied. Thus it will still take $\Theta(n)$ time. Now, for every Max-Heapify call, one of the smallest values in the heap is kept at the top of the heap. Therefore, it will take $O(\log n)$ to move the smallest value to the bottom of the heap. This will take time equal to the depth of the heap which is $O(\log n)$

∴ Running time of HEAPSORT when array is in decreasing array is $\Theta(n \log n)$

When the array A is in increasing order, the array will be first converted to MAX-HEAP by BUILD-MAX-HEAP call, which will take more comparisons than in decreasing order, but differs by only a constant factor. Therefore

Assignment 2

Chitrarth Singh
A20387080

2 continue and 3.

BUILD-MAX-HEAP will still take $\Theta(n)$ time. MAX-HEAPIFY will take worst time equal to the lower bound of $\log n$. Therefore it will be $\Omega \log n$.
∴ Running in increasing order $\Rightarrow \Omega(n \log n)$

3

(a). Tail Recursive Quicksort (TRQ) does the same partitioning like Quicksort. The only difference is that TRQ calls itself by setting $p \leftarrow q+1$. Therefore TRQ correctly sorts the array A. as it employs the same operations as the traditional Quicksort algo but without the second recursive call.

(b). When Tail Recursive Quicksort is called on an array in increasing order, the partition function will partition the element such that the first $(n-1)$ element falls in the left subarray and then the left subarray is sorted recursively by calling the TRQ function with each recursive call, left subarray length is decreased by one which equals $O(n)$ calls. Therefore stack depth will be $\Theta(n)$

Assignment 2

Chitrarth Singh
A20387080

3 continue:

(d) Tail Recursive Quicksort Recurrance relation:

$$f(n) = 1 + \frac{1}{n} \sum_{i=1}^{n-1} S(i).$$

Assume initial values to be $t_0, t_1, t_2 \cdots t_n$ for $n_{i=0}$

$\therefore$ we can rewrite it as:

$$t_n = a + \frac{1}{n} \sum_{i=1}^{n-1} t_i \quad \text{where } n > n_0 \quad -(1)$$

Now lets substitute $n-1$ for n

$\therefore$ we get,

$$(n-1)\, t_{n-1} = a(n-1) + \sum_{i=1}^{n-2} t_i \quad -(2)$$

Subtract (2) from (1)

$$\Rightarrow n t_n - (n-1) t_{n-1} = an - a(n-1) + t_{n-1}$$
$$\Rightarrow n t_n - n t_{n-1} + t_{n-1} = an - an + a + t_{n-1}$$
$$\Rightarrow n t_n - n t_{n-1} = a$$
$$\Rightarrow t_n - t_{n-1} = a/n$$

$$\therefore \sum_{i=1}^{n} t_n - t_{n-1} = \sum_{i}^{n} a/i = a \ln n + O(1)$$

i.e. $\Rightarrow \underline{O(\log n)}$ $\{\because t_n = a \ln n + O(1)\}$

Assignment 2

Chitrarth Singh
A20387080

4.

4. Problem 8.3(a).

The problem can be solved using Radix sort. Let suppose the numbers have at most 'x' digits. The sorting procedure will take place according to the i^{th} digit of each number from $i = 1$ to x.

Let d_i be the number of integers having i^{th} digit. Therefore the sorting algorithm will take time which is probably less than $\sum_{i=1}^{x} i \cdot c(d_i + D)$ time (C & D are constants.)

$$\Rightarrow \sum_{i=1}^{x} i \cdot c(d_i + D)$$

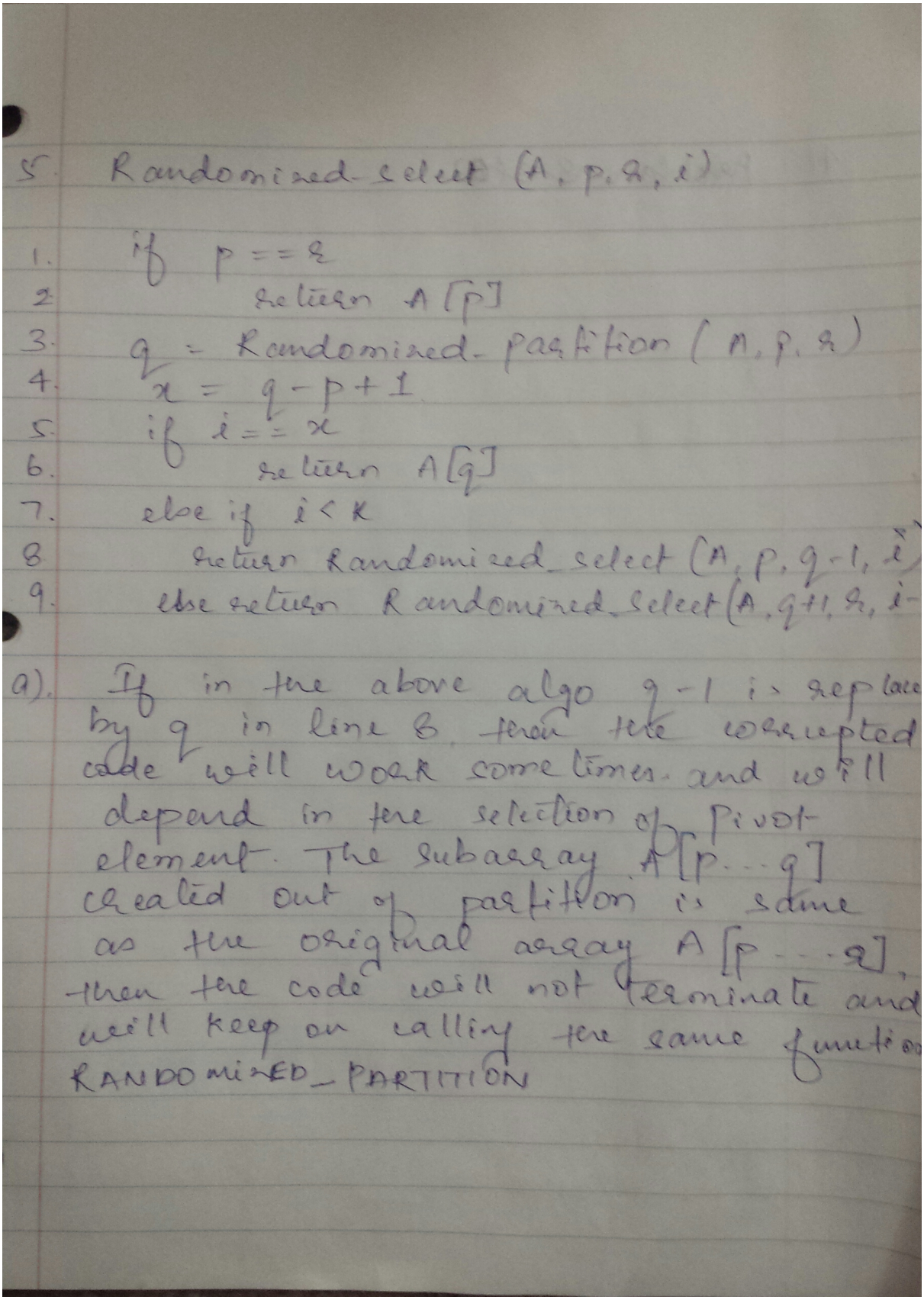
$$= c \sum_{i=1}^{x} i d_i + D \sum_{i=1}^{x} i \leq (C+D)n.$$

$$= O(n).$$

Assignment 2

Chitrarth Singh
A20387080

5.



5. Randomized-select (A, p, q, i)

```
1.  if p == q
2.      return A[p]
3.  q = Randomized-partition (A, p, q)
4.  x = q - p + 1
5.  if i == x
6.      return A[q]
7.  else if i < k
8.      return Randomized_select (A, p, q-1, i)
9.  else return Randomized_Select (A, q+1, q, i-
```

a). If in the above algo q-1 is replaced by q in line 8, then the corrupted code will work sometimes and will depend in the selection of Pivot element. The subarray A[p...q] created out of partition is same as the original array A[p...q], then the code will not terminate and will keep on calling the same function RANDOMIZED_PARTITION

Assignment 2

Chitrarth Singh
A20387080

5 continue:

(b). The worst case running time will be infinite because the code will never terminate.

(c). Best case:- Occurs when the i^{th} element is selected as the pivot element during the first partition phase. Therefore the recursive calls will give the order of algorithm to be $\Theta(n)$. That is also the time of Randomized-Partition.

(d). Average case running time:

$$T(n) \leq \sum_{k=1}^{n} x_k \cdot T(\max(k, n-k)) + O(n)$$

$$\max(k, n-k) = \begin{cases} k & \text{if } k \geq \lceil n/2 \rceil \\ n-k & \text{if } k < \lceil n/2 \rceil \end{cases}$$

subtract (2) from (1), we get:-

$$E[T(n)]\left(1-\frac{1}{n}\right) \leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} E[T(k)] + \alpha$$

$$E[T(n)]\left(1-\frac{1}{n}\right) \leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} ck + an.$$

$\therefore E[T(n)] \leq cn$ for some const. c.

Assignment 2

Chitrarth Singh
A20387080

5 continue:

$$E[T(n)](1-1/n) \le \frac{2}{n}\sum_{k=\lceil n/2\rceil}^{n-1} ck + an.$$

$$= \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil n/2\rceil - 1} k\right) + an$$

$$= \frac{2c}{n}\left(\frac{n(n-1)}{2} - \frac{\lceil n/2\rceil \, n/2 - 1}{2}\right)$$

$$\le \frac{2c}{n}\left(\frac{n(n-1)}{2} - \frac{(n/2)\cdot(n/2-1)}{2}\right) + an$$

$$= c\left(\frac{3n}{4} - \frac{1}{2}\right) + an.$$

∴ For sufficiently large n

$$E[T(n)] = O(n).$$

e). The change from above observation is that the average case running time is still $O(n)$. The pivot element is selected randomly across n elements which gives linear growth with respect to growth of data elements in array. The probability of selecting the same element as pivot is very low. Therefore running time will still be $O(n)$