Problem 3

part a)

1. Analysing running time for all parts :

```
if (right <= left + 10)
{
    InsertionSort (a, left, right);
    return a[k + left];
}
```

⇒ Although the worst case runtime for insertionsort is $O(n^2)$, the input size will always smaller than 10, in this case, the runtime for this if statement is $O(1)$

```
else {
    int smallsize = (right - left) / 5;
    T *b = new T[smallsize]
```

⇒ constant runtime : $O(1)$

```
for (int i=0; i< smallsize; i++)
    b[i] = quantile(a, left+5*i, left+5*(i+1)-1, 2);
```

The input size for each recursive call is $5(i+1) - 5i = 5$, so the quantile() will go into the if statement we analysed before, which takes constant time $O(1)$

$\because$ smallsize $= n/5$

$\therefore \frac{n}{5}$ times of recursive calls are called

$\therefore$ total runtime $= \frac{n}{5} \cdot O(1) = O(n)$.

```
T pivot = quantile(b, 0, smallsize-1, smallsize/2)
```

⇒ The array has a size of $\frac{n}{5}$

$\therefore$ The recursive call will take $T(n/5)$.

```
int p = linearSearch(a, left, right, pivot);
a.swap(p, right);
```

⇒ worstcase runtime $O(n)$

⇒ constant time $O(1)$

```
int m = partition(a, left, right);
```

⇒ partition will take $O(n)$

```
if (left+k==m) return a[m]
else if (left+k<m) return quantile(a, left, m+1, k);
else return quantile(a, m+1, right, k-(m+1-left));
}
}
```

Because the size of b[i] is $n/5$ and the pivot is the median of this array, so $\frac{n}{5} \cdot \frac{1}{2} = \frac{n}{10}$ elements have to be smaller than pivot. Because these $\frac{n}{10}$ elements are also medians in its 5-element sub array, so the elements on the left side in the sub-arrays also have to be smaller than pivot.

$\therefore$ at least smaller $= \frac{n}{5} \cdot \frac{1}{2} + \frac{n}{5} \cdot \frac{1}{2} \cdot 2 = \frac{n}{10} + \frac{n}{5} = \frac{3n}{10}$

$\therefore$ at least $\frac{3n}{10}$ elements in a are smaller than pivot.

By using the same theory for the elements larger than pivot.

at least larger $= \frac{n}{5} \cdot \frac{1}{2} + \frac{n}{5} \cdot \frac{7}{2} = \frac{3n}{10}$

$\therefore$ at least $\frac{3n}{10}$ elements in a are larger than pivot

Therefore, if at least $\frac{3n}{10}$ elements for smaller and larger than pivot

$n - \frac{3}{10}n = \frac{7}{10}n$

$\therefore$ At most $\frac{7}{10}n$ elements larger and smaller than pivot.

$\therefore$ At most $\frac{7}{10}n$ element the array will have for last or second-to-last time.

$\therefore$ for last or second-to-last time $\Rightarrow T(7n/10)$

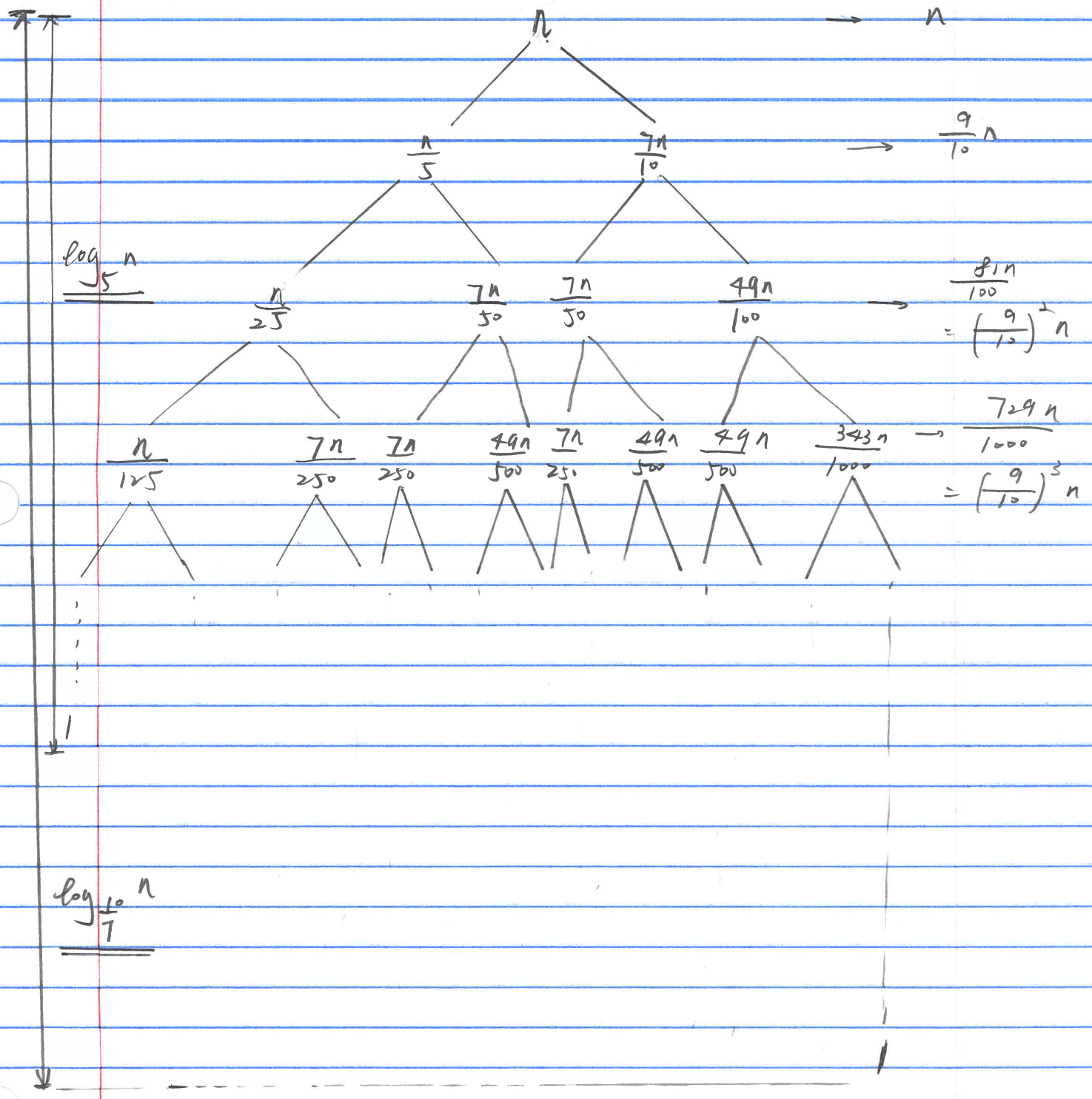
$\therefore$ when $n \geq 10$

$$T(n) = O(1) + O(n) + T(n/5) + O(n) + O(1) + O(n) + T(7n/10)$$
$$= T(n/5) + T(7n/10) + O(n)$$

In addition: The base case is when $n < 10$, it will just return median using insertion sort which will take $O(1)$ with size smaller than 10

Question 3
part b.

Recursive call tree (The value on each node represents both size of array and run time)



- For each layer, if you sum up the size of each node
$n,\ \frac{9}{10}n,\ \left(\frac{9}{10}\right)^2 n,\ \left(\frac{9}{10}\right)^3 n \ \cdots\ \left(\frac{9}{10}\right)^i (n)$

The reason for that is $\frac{1}{5} + \frac{7}{10} = \frac{9}{10}$. Each new layer will simply time $(\frac{1}{5} + \frac{7}{10})$ for each leaf in the previous tree.

$\therefore$ The total work $= \left(\frac{9}{10}\right)^i n$

- Because $\frac{1}{5} \neq \frac{7}{10}$, the tree will have different height on each leaf, However, the largest height will happen on the right most leaf, because $\frac{7}{10} > \frac{1}{5}$, it will take more layers to get to size = 1.

$\therefore$ number of layers $= \log_{\frac{10}{7}} n$

- $\because \frac{1}{5} < \frac{7}{10}$

$\therefore$ The smaller size recursion will always on the left, the larger size recursion will always on the right.

$\therefore$ The leaf with smallest height is the left most path, smallest number of layer $= \log_5 n$

Therefore, in each the first $\log_5 n$ layer, there are $\left(\frac{9}{10}\right)^i n$ nodes in total, so it's a geometric progression

For the rest, each layer has fewer than $\left(\frac{9}{10}\right)^i n$, so every layer is at most $\left(\frac{9}{10}\right)^i n$, so we can use the at-most value to calculate the upper-bound ☺

$$T(n) = n + \frac{9}{10}n + \left(\frac{9}{10}\right)^2 n - \cdots \left(\frac{9}{10}\right)^i n$$

$$= \sum_{i=0}^{\log_{\frac{10}{9}} 10n - 1} \left(\frac{9}{10}\right)^i n$$

$$= \frac{1 - \left(\frac{9}{10}\right)^{\left(\log_{\frac{10}{9}} n - 1\right)}}{1 - \frac{9}{10}} n$$

we can bound this equation :

$$T(n) = \frac{1}{\frac{1}{10}} n$$

$$= 10n$$

$$\therefore T(n) = O(10n)$$

$$= O(n)$$