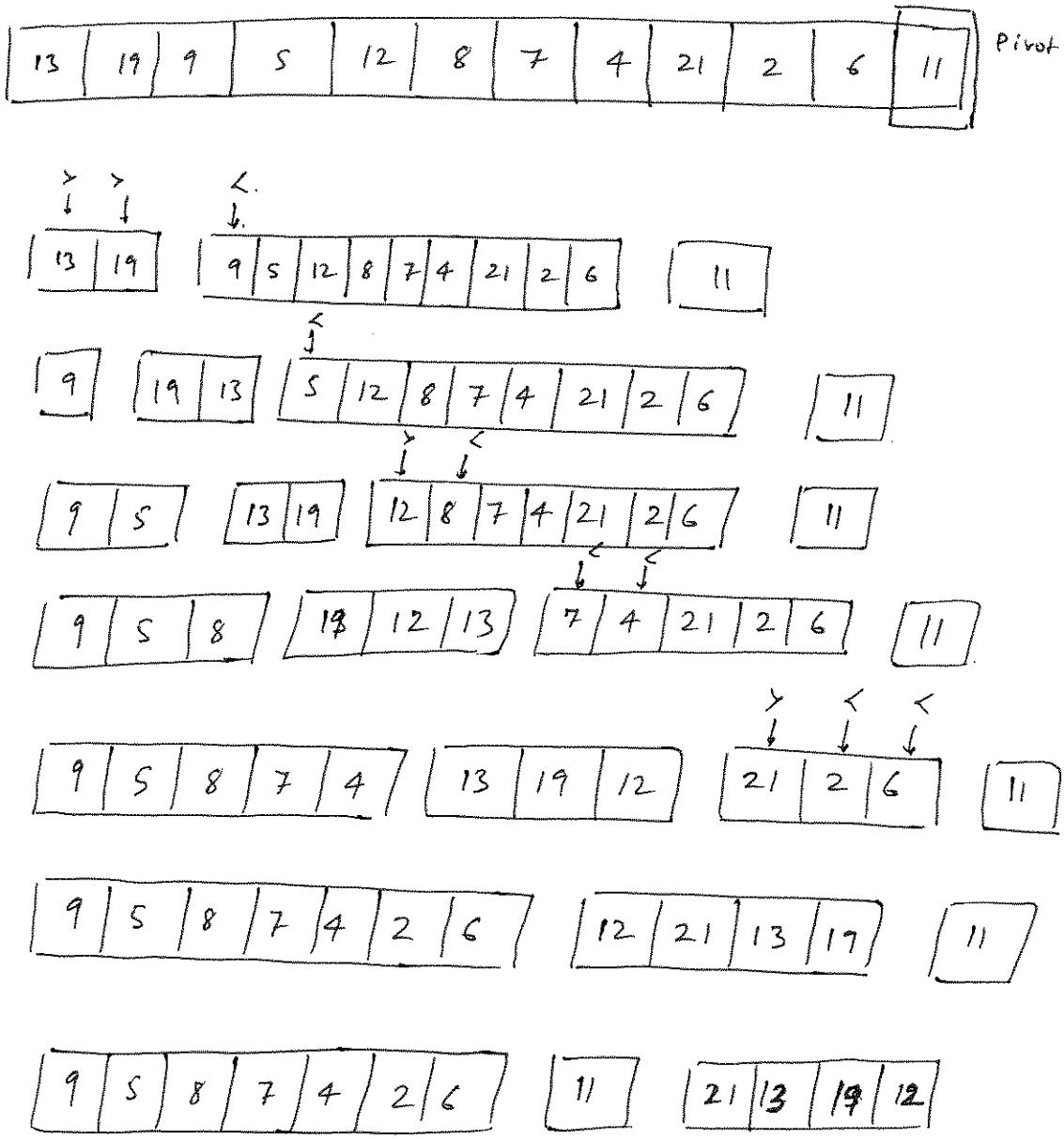
CHAPTER 7

Qn 7.1-1

$A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11 \rangle$.



Qn 7.1-2

```
Partition (A, p, r).
1.  x = A[r]
2.  i = p - 1
3.  for j = p to r - 1
4.      if A[j] ≤ x.  A[j] < x
5.          i = i + 1
6.          exchange A[i] with A[j]
7.  if i = p - 1 then
8.          return ⌊(p+r)/2⌋
9.  exchange A[i+1] and A[r]
10. return (i+1).
```

CHAPTER 7

Qn 7.2-1

$$T(n) = T(n-1) + \Theta(n).$$

$$\leq c\sum_{i=0}^{n-1}(n-i) + \Theta(n).$$

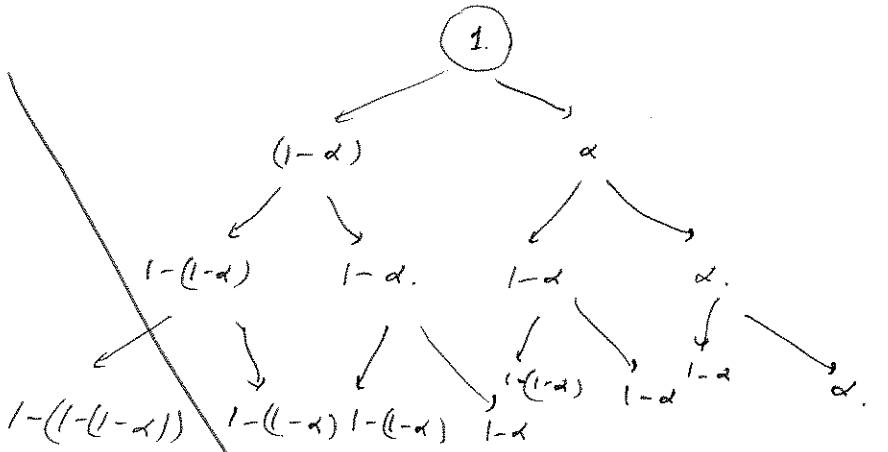
$$\leq \Theta(n^2) + \Theta(n)$$

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

Qn 7.2-2 when all elements of array A have same value.

$\Theta(n^2)$ from $T(n) = T(n-1) + \Theta(n)$

Qn 7.2-5



$1-(-(1-(1-(1-\alpha))))$

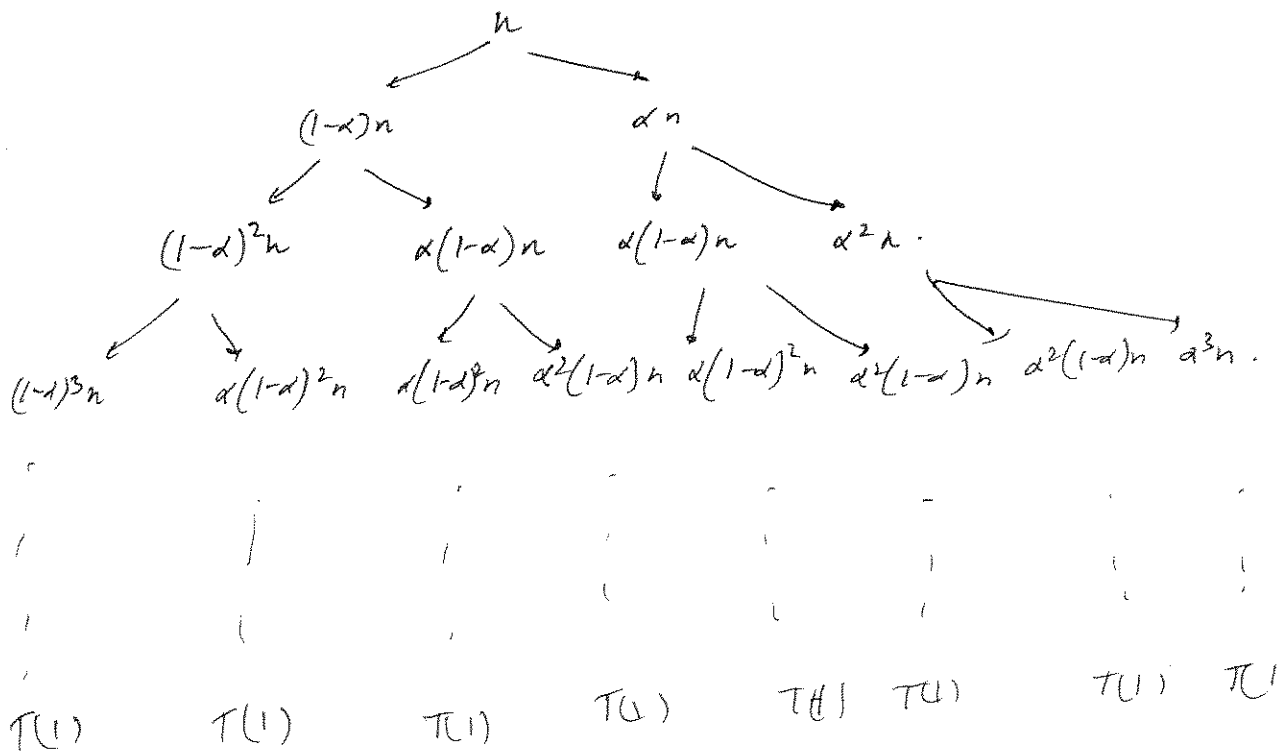
$1-(\underbrace{\ldots(1-(1-\alpha))}_{\log \alpha \text{ when } \alpha = 0}) = 1$

P.T.O

Qn 7.2-5

Splits in proportion $(1-\alpha)$ & α.

$$T(n) = T((1-\alpha)n) + T(\alpha n) + C.$$



Min height of tree $= \log_{(1/\alpha)} n$

$$= \frac{\log n}{\log(1/\alpha)}.$$

$$= -\frac{\log n}{\log \alpha}.$$

{ Since $0 < \alpha \leq 1/2$ }

So at $\alpha = 1/2$ the division remains same

$\Rightarrow$ min ht = max ht

$\Rightarrow$ no difference

or for min

$\alpha < (1-\alpha)$.

$\Rightarrow$ $\alpha \to$ min

Max height of tree $= \log_{\frac{1}{(1-\alpha)}} n$

$$= -\frac{\log n}{\log(1-\alpha)}$$

Qn 7.2-6

$0 < \alpha \leq 1/2$

$\Rightarrow \quad 2.0 < 2\alpha \leq 1$

$\Rightarrow \quad 0 > -2\alpha \geq -1$

$\Rightarrow \quad 1-0 > 1-2\alpha \geq 1-1$

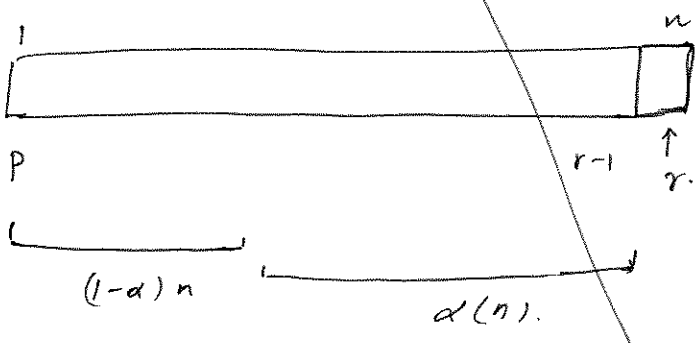
$\Rightarrow \quad \boxed{1 > 1-2\alpha \geq 0}$?

$T(n) = T(n-\alpha) + T(\alpha) + c.$

$\alpha = 1 \quad = T(n-1) + T(1) + c.$

$\alpha = 2 \quad = T(n-2) + T(2) + c.$

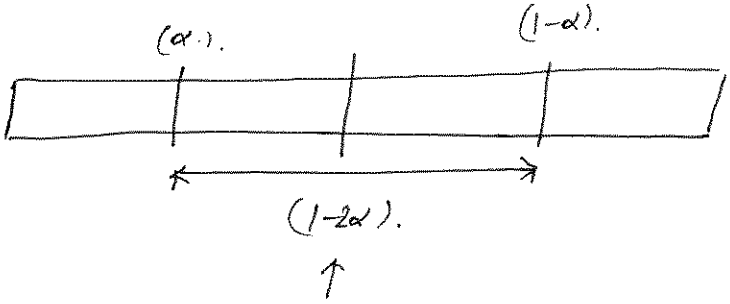
$\alpha = \frac{1}{2}n \quad = T(n-\frac{1}{2}n) + T(n/2) + c. = T(n/2) + T(n/2) + c.$



P.T.O

Q: 7.2-6 $0 < \alpha \leq 1/2$

PARTITION produces random split. with prob $1/n$ for each rank of pivot.



↑

Amount of region where the PARTITION split will be more balanced than $(1-\alpha)$ and α.

Qn 7.4-1

$$T(n) = \max_{0 \le q \le n-1} \left(T(q) + T(n-q-1) \right) + \Theta(n) = \Omega(n^2) \; ?$$

$$= \max_{0 \le q \le n-1} \left(q^2 + (n-q-1)^2 \right) + \Theta(n).$$

$$= \max_{0 \le q \le n-1} \left(\sum_{q=0}^{n-1} T(q) + \sum_{q=0}^{n-1} T(n-q-1)^* \right) + \Theta(n).$$

$$= \max_{0 \le q \le n-1} \left((q^2) + (n-q-1)^2 \right) + \Theta(n).$$

1/n	$q=0$	$=$	$(0) + (n-1)^2 + \Theta(n).$
1/n	$q=1$	$=$	$(1)^2 + (n-2)^2 + \Theta(n).$
	⋮		⋮
1/n	$q=n-1$	$=$	$(n-1)^2 + (0)^2 + \Theta(n).$

Sum

$$\frac{1}{n} \sum_{i=0}^{n-1} (n-i)^2 + \frac{1}{n} \sum_{i=0}^{n-1} \Theta(n).$$

$$= \frac{2}{n} \Theta(n(n)^2) + \Theta\left(\frac{n\,\Theta(n)}{n}\right).$$

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

$$= \underline{\Theta(n^2)}$$

Qn 7.4-2

Best-case → when both partitions are equal.

$\Rightarrow \quad T(n) = 2\,T(n/2) + \Theta(n).$

$\Rightarrow \quad T(n) = \Theta(n \log n)$

Qn 7.4-3

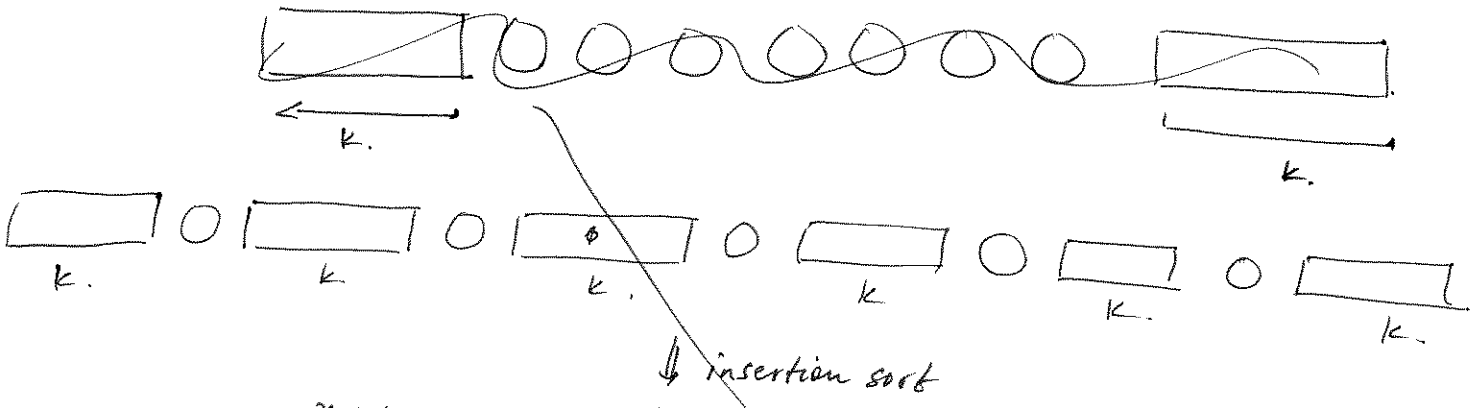
$q^2 + (n-q-1)^2$ is max when $q = 0$ or $n-1$.

Diff w.r.t. q

$$\frac{\partial}{\partial q} q^2 + \frac{\partial}{\partial q} (n-q-1)^2 = 0$$

$$\Rightarrow \quad 2q + 2(n-q-1) \cdot (-1) = 0.$$

Qn 7.4-5.



↓ insertion sort runs over n elements.
Sorts only k elements of each partition.

P.T.O

Qn 7.4-5

We can argue that every element in the array is (at max) within k places in the array. at max. Hence, the insertion sort is going to take $O(nk)$ time to sort the array since it only needs to check max k places for each element.

Quicksort was taking an expected time of $O(n \log n)$ but now returns k elements, i.e. it does not divide further if k elements are reached. Hence, the quick sort procedure will now take $O(n \log(n/k))$

Hence $O(nk + n\log(n/k))$

In theory: $nk \leq n\log(n/k)$

$\Rightarrow k \leq \log(n/k)$

$\Rightarrow \boxed{2^k \leq \frac{n}{k}}$

$\Rightarrow k + \log(k) \leq \log(n)$

$\Rightarrow \boxed{k2^k \leq n}$ Must follow this condition.

Qn 7-2 (b)

```
PARTITION(p, r)   //Partition A[p...r] to A[p...q-1]
                                          A[q...t]
                                          A[t+1,...r]
1.  x = A[r]
2.  i = p-1
3.  j = p-1
4.  for k = p to r-1
5.      if A[k] <= x.
6.          j = j+1
7.          exchange A[j] with A[k]
8.      else if A[k] < x.
9.          i = i+1
10.         j = j+1
11.         exchange A[j] with A[k]
12.         exchange A[i] with A[j]
13. exchange A[j+1] with A[r]
14. return (i+1, j+1).
```

Q7.4

(a)

The quicksort algorithm runs the first recursive part and then the next recursive part.

The tail-recursive quicksort runs the first recursive part then moves the problem to the second half and runs the second half as a new recursive part, thus the same.

(b) The stack depth is $O(n)$ when the partition returns n, i.e. the complete array leaving out the pivot will take part in the recursive call.

(c) Use RANDOMIZED-PARTITION instead of PARTITION!

On 7-6

* FuzzySort(intervals, p, r)

1. if (p < r) then
2. q = partition (intervals, p, r)
3. fuzzysort(intervals, p, q-1)
4. fuzzysort(intervals, q+1, r)

(*) When all intervals overlap:

⇒ For n times it will check.

or $T(n) = T(n-1) + O(n)$

by subs^n : $\Theta(n)$.

Partition (intervals, p, r)

q = random (p, r)
j = p - 1
for i = p to r - 1
  if ~~(interval~~
  low = ~~low - int(intervals)~~ intervals[i][0]
  high = interval[i][intervals[i].length]
  if low ~~<~~ > A

* partition(~~intervals~~ A, p, r)

1. pivot = random (p, r)
2. exchange A[pivot] and A[r]
3. pivot = A[r]
4. i = p - 1
5. for j = p to r - 1
     if ~~low~~ high (A, j) <= ~~pivot~~ low (A, r)
6.   exchange A[j] and ~~pivot~~A[i]
7.
8.   i++
9. exchange A[i+1] and A[r]
10. return i+1.

* high (A, j)
1. return A[j][A[j].length]

* low (A, j)
1. return A[j][0]