Assignment - 3

Answer to the question no. - 1

(a) Random variable is a variable whose possible values are numerical outcomes of a random phenomenon. For the given problem, let X be the outcome of a single roll of a fair 3 face die.

So, X = {1, 2, 3, 4, 5, 6} and Probability for each value of X will be equal and that will be 1/6.

We know, Expected value, E[X] = $\sum_{x_i} x_i * P(x_i)$

= 1 * 1/6 + 2 * 1/6 + 3 * 1/6 + 4 * 1/6 + 5 * 1/6 + 6 * 1/6

= 7/2

(b) Let, X_i be the indicator random variable counting when face i is up, where i = 1 to 6 and X be the random variable for sum of n die rolls.

Now, Expected value, E[X] = $\sum_{i=1}^{6} E[x_i] * i$

= $\sum_{i=1}^{6} \frac{n}{6} * i \; ; \{E[x_i] = \frac{n}{6}\}$

= n/6 * $\sum_{i=1}^{6} i$

= n/6 * {6 * (6+1)}/2

= n * 7/2

Answer to the question no. - 2

To denote the gain, Let us take a random variable X. So, we can write,

If we roll two 1's, gain will be X(1-1) = 10

If we roll one 1, gain will be X(x-1) = 1

If we roll no 1, gain will be X(x-x) = -0.5

Probability for each gain,

P(X = 10) = 1/6 * 1/6 = 1/36

P(X = 1) = 2 * 1/6 * 5/6 = 10/36

P(X = -0.5) = 5/6 * 5/6 = 25/36

So, Expected value of the game, E(X) = P(X = 10) * 10 + P(X = 1) * 1 + P(X = -0.5) * (-0.5)

= 1/36 * 10 + 10/36 * 1 + 25/36 * (-0.5)

= 10/36 + 10/36 – 25/72

= 5/24

Answer to the question no. - 3

In randomized quicksort, the random number generator is used to select a random pivot for each partitioning step. The number of partitioning steps required, which in turn depends on how the elements are divided in each step, determines how many calls to the random number generator are necessary.

(a) Best Case: the pivot divides the array perfectly in half each time. The recursion tree is balanced, with a height of logn. At each level ^{2k} pivots are selected for k = 0 to logn – 1. So,
Total number of pivots = $\sum_{k=0}^{logn-1} 2^k = 2^{logn} - 1 = n - 1$
Since a random pivot is selected at each partition call, the number of calls to the random number generator can be expressed as,

$$C(n) = 2\,C(n/2) + 1 = \theta(n)$$

(b) Worst Case: the pivot divides the array such that one part has 0 element and the other part have the rest of the elements. So, at each step only one pivot is selected and then each pivot reduces the problem size by 1 that makes the number of steps equal to n-1.
Since a random pivot is selected at each partition call, the number of calls to the random number generator can be expressed as,

$$C(n) = C(n-1) + 1 = \theta(n)$$

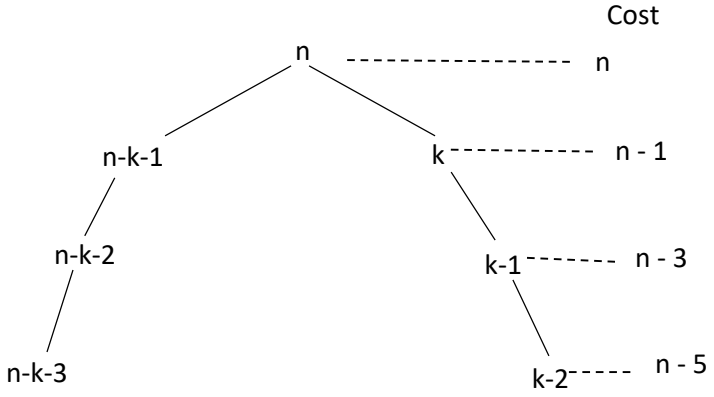
(c) Average Case: the pivot tends to divide the array into reasonably balanced parts. The expected number of comparisons is O(nlogn) in average case. However, the number of pivot selection remains n-1. From best and worst case complexity of random number generator calls, we can conclude that average case should be $\theta(n)$.

Answer to the question no. - 4

(a) With n equal keys each time the function partition (A, p, q) is called, it will return q because the loop condition A[j] <= x will always be true. If there were n element in the original array, then a subproblem with n-1 element will be generated.
So, we can write T(n) = T(n-1) + n = $\theta(n^2)$
In the same arrangement, randomized quicksort will select pivot randomly but the partition routine still returns a n-1 and 0 partition. So the complexity remains same $\theta(n^2)$.

(b) In (a), the partition routine returns left partition (B) with size n-1 and right partition (C) with size 0, if the original sub problem has size n.
If we change A[j] <= x to A[j] < x in the pseudo code for partition, then left partition (B) with size 0 and right partition (C) with size n-1 will be created. So, we can write T(n) = T(n-1) + 1 = $\theta(n^2)$.

(c) In deterministic quicksort with two distinct keys, the smaller number can be selected as pivot or the larger one. But in either case that will be two subproblem with same keys after first partition. So the recursion tree will have the height of max (n-k-1, k) where k = number of element in one subarray.



So, runtime = $\sum_{k=0}^{\max(n-k-1,k)} n-(2k+1) \leq \sum_{k=0}^{n} n-(2k+1) = O(n^2)$

(d) Three way partition (A, l, r) {

```
pivot = A[l]
low = l
high = r
i = l
while(i<=high):
        if A[i] < pivot:
                swap (A[low], A[i])
                low ++
                i ++
        else if A[i] > pivot:
```

```
                    swap (A[i], A[hight])
                    high --
            else:
                    i++

    return low, high;

}
```

(e) In the worst case, only one element of a distinct key is present in the array and we choose the other distinct key as pivot. Suppose, we have an array [a, b, b, a, b, a, b, a] and we take "a" as pivot initially. After partitioning we will get [a, a, a, a, b, b, b, b] and as we process all n elements, the cost will be O(n). If we consider "a" as the smallest value, then there is no value less that "a" and b is greater than a. So we process the remaining elements [b, b, b, b] here the first element "b" is chosen as pivot and no element greater or less than "b" is in this subarray. So, no further recursion is needed since there are no elements less than or greater than the pivot. After that the array is fully partitioned and all elements equal to "a" and "b" are grouped together. So, the runtime is O(n) in worst case.

(f) In the worst case we have 'd' height of the tree. each time the partition routine is called all the element with some particular key which is selected as pivot will no longer exist in the subproblem. Worst case will happen when we have only one element for d-1 keys and (n-d+1) elements have the same key. We select keys with single element as pivot each time.

So, at each level, one distinct key that is pivot, is eliminated from further consideration. The remaining elements belong to fewer that d keys. Since there are d distinct keys, the maximum depth of recursion is O(d) and each recursive call reduces the number of distinct keys by at least one. At each level of recursion, all elements are processed once and the total number of elements processed at each level is n.

Total elements processed per level = n, time per level = O(n) and number of levels = d.

So, Total time = $\sum_{k=0}^{\mathrm{d}} O(n) = O(nd)$