**LAB 3**

1. Suppose an array of length n is populated randomly with the letters A and B, with the guarantee that both letters occur at least once and that A and B are equally likely to occur. Let Search(S, x) be the usual algorithm for searching for an element x in an array S, which returns the position of the first occurrence of x in S (or –1 if not found). Prove that the average-case running time for Search(S,x) is < 2 whenever x is either A or B, and therefore Search (for A or for B) has average case asymptotic running time O(1).

**Solution**:

**Best case:** x is found at first position. Running time is 1, i.e., O(1).

**Worst case:** x is located in the last slot. Running time is n, i.e., O(n)

**Average case:** Let Y be a random variable denoting the number of slots of the array we have to check to find the first occurrence of x. Since the probability of finding x (either A or B) at any given slot is 1/2, we have:

Pr(Y = i) = (1/2)i Where i is the slot position first instance of x is found

E[Y] =

If N approaches , E[Y] = 2. For any N less than then E[Y] < 2.

Therefore, the asymptotic running time of the algorithm is O(1).

1. A company uses a well-known sorting algorithm to sort its data. A best case for this sorting algorithm occurs when its input is an already-sorted array. In such cases, it runs in O(n) time. A worst case occurs when its input is reverse-sorted. In that case, it runs in O(n2 ) time. The company knows from experience that all input arrays are either sorted or reverse sorted, but nearly all input arrays are already sorted. In fact, it is estimated that, for any collection of n arrays from the pool of all length-n arrays in the company’s data store, only one of these arrays is ever reverse-sorted. What is the average-case asymptotic running time of the algorithm, given this distribution of inputs? Prove your answer. Hint: Review the Lesson 3 slides.

**Solution**:

**Best case:** the input array is always sorted. Running time is O(n).

**Worst case:** the input array is always reverse sorted. Running time is O(n2)

**Average case:** Let Y be a random variable denoting the running time of an input array as length of the array. When the input array of length n is sorted, y1=n. When the input array of length n is reverse sorted, y2 = n2. The input array is either sorted or reverse sorted. Only one of such n arrays is reverse sorted.

E[Y] = y1Pr(Y= y1) + y2Pr(Y= y2)

= n( + n2( = 2n

So, the asymptotic running time is O(n).

1. A die is tossed repeatedly.
   1. What is the expected number of tosses required to get a 6?

**Answer**: If we consider getting a 6 as success and any other number as failure, then we have a Bernoulli trial experiment, where probability of success is 1/6 and probability of failure is 5/6. Each experiment is independent. In such cases, the number of trials required to get the first success is given by:

Consider Y a random variable which is number of tosses required to get 6.

E(Y) = 1/p = (1 % (1/6)) = 6

* 1. What is the expected number of tosses required to get a total of three 6’s? In each case, prove your answer:

**Answer**: Extending from (a), we now need to get two success. In such cases, the expected number of failures before we get 2 6s is given by:

E[X] = , where k is number of successful outcomes required (2 in this case), and p is the probability of success (1/6) in this case.

E[X] = 10

So in total we need to make X + k = 10 + 2 = 12 attempts to get 2 6s.

1. Design an algorithm that does the following: Input is a set S of n integers together with an integer k. Your algorithm outputs “true” if there is some subset of S, the sum of whose elements is exactly k; it outputs “false” if no such subset can be found. What is the asymptotic running time of your algorithm? Explain.

**Solution**:

For this problem, I’ll base my solution on the powerSet algorithm which was described on question 6 of lab 2. The PoserSet algorithm gives a set consisting of all subsets of a given set.

**Algorithm**: CheckSubsetSum(S, n, k)

***Input***: A set of integers with n number of elements

***Output***: It gives “true” if sum of any of the subsets = k. False otherwise

**if** n <= 0 **then**

**return** false

P 🡨 PoserSet(s) //Let the running time of PowerSet algorithm be Tp

**for each** x **in** P **do //**runs for number of elements in set P

sum 🡨 0

**for each** i **in** x do //average size of subsets is n/2 elements

sum 🡨 sum + i

**if** sum = k **then**

**return** true;

**return** false;

**Algorithm**: PowerSet(X)

***Input***: A list X of elements

***Output***: A list P consisting of all subsets of X – elements of P are *Sets*

P ← new list

S ← new Set //S is the empty set

P.add(S) //P is now the set { S }

T ← new Set

**while** (!X.isEmpty() ) **do**

f ← X.removeFirst()

**for each** x **in** P **do //**each iteration doubles size of P

T ← x U {f} // T is the set containing f & all elements of x

P.add(T)

**return** P

To find running time of the algorithm in its entirety, we begin our analysis with running time of PowerSet algorithm. The running time of the inner for loop in the PowerSet algorithm doubles each time an additional element is removed from the original set in the outer while loop. Assuming the logic inside the inner for loop takes constant running time of c, we have the running time of the PowerSet algorithm as:

Tp = = ≈ c //Using formula in Math review slide

Worst case scenario happens when we don’t get any subset with sum equal to k. The worst case running time T(n) of the algorithm then can be estimated by:

T(n) = + c0

Therefore, the asymptotic running time is: O(

1. Goofy’s method of array sorting:

Solution:

Goofy’s method will obviously work as it checks every possible ordering of elements of the array. One of them will give the correctly sorted array.

**Best case:** the input array is already sorted, so O(1).

**Worst case:** sorted order comes last: i.e., O(n!)

**Average case:** Let Y be a random variable denoting the running time after each permutation. We have a total of n! permutations each having a probability of 1/n! in being the correctly sorted (assuming that each permutation is equally likely to be sorted one).

E[Y] = =