Question 1:

First assume input is uniformly distributed. More precisely it is $(n+1)/2$. When you search for a particular element x in an array of size n, that element may be located at the position either 1, or 2, ... or n. When we search we check each element in the array and compare it with x, and so when we reach k^{th} element of the array we have already done k comparisons.

If it is at 1 then you find it in 1 comparison, if it is at 2, you find it in 2 comparisons, ..., if it is at n then you do n comparison in order to find it. In order to average it you sum the total number of comparisons $1+2+\cdots+n= ((n+1)n)/2$ and divide it by n (size of the array) resulting in $(n+1)/2$.

Question 2:

Solution: Let X be a random variable denoting the running time of the algorithm on any given input array that could occur. Since already sorted arrays run in Θ(n) time, there is a constant c and a function f such that running time for such arrays is asymptotically equal to f(n) = c ·n for some c. Likewise, for reverse sorted arrays, there is a constant d and a function g such that running time to sort these arrays is asymptotically equal to g(n) = d · n2.

Question 3. A die is tossed repeatedly.

a. What is the expected number of tosses required to get a 6?

Ans

We have a 1/6 probability of rolling a 6 right away, and a 5/6 chance of rolling something else and starting the process over (but with one additional roll under your belt).

Let E be the expected number of rolls before getting a 6; by the reasoning above, we have:

E=(1)(1/6)+(E+1)(5/6)

Solving for E yields E=6.

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

Ans:

Solving for 3E yields 3E=6 *3=18.

Question 4. **Algorithm: SubsetSum(S, k)**

Input: A set S of n integers

Output: true if the sum of the elements of some subset of S is k; false otherwise.

//using PowerSet algorithm from Lab 2

while T in P do

accum 0

for x in T do

accum = accum + x //accum is the sum of the elements of T

if accum = k then

return true

return false

Analysis: The call to PowerSet(S) is . The while loop that scans the 2n subsets of in P and does n work on each requires . So complexity is $O(n^2)$

Question 5 Goofy Sort. Goofy has thought of a new way to sort an array arr of n distinct integers:

(a) Step 1: Check if arr is sorted. If so, return arr.

(b) Step 2: Randomly arrange the elements of arr (using your work in Problem 2 of this lab)

(c) Step 3: Repeat Steps 1 and 2 until there is a return.

Answer the following:

(A) Will Goofy's sorting procedure work at all? Solution: Yes, most of the time (see analysis below).

(B) What is a best case for GoofySort? Solution: The best case is when the input array is already sorted.

(C) What is the running time in the best case? Solution: The running time in the best case is O(n) (it takes O(n) time to verify that arr is in sorted order). (D) What is the worst-case running time? Solution: ∞– this happens if no random arrangement ever occurs in sorted order.

(E) Is the algorithm inversion-bound? Solution: No. Consider the case in which the input array is in reverse-sorted order (so there are $n(n-1)/2$ inversions) and, after the first trial, the randomly generated array produced is in sorted order. In that case, generating the array required n comparisons and checking array is sorted required approximately n more comparisons; but $n + n < n(n-1)/2$, so, in this case, fewer comparisons are done than there are inversions in the input array.

Question 6: Answer in the source code file