Matt Ronan 2/19/20

Algorithms Assignment 3

Section 1 Question 1:

English: To find the number of times a specific number appears in array we use binary search to find each occurrence of the number. We start by checking the numbers on one side of the middle number and searching for its first and last occurrence using binary search. Then we do the same on the other side and add the total

Pseudocode:

count(double[] A, double x)

return rightSide(A, x) - leftSide(A, x) + 1;

end count

leftSide(A[], x) {

start = 0; end = A.length - 1; mid = 0;

while(start <= end) {

mid = (end + start)/2;

if((mid == 0 && A[mid] == x) || (A[mid] == x && A[mid - 1] < x))

return mid;

else if(A[mid] < x)

start = mid + 1;

else if(A[mid] >= x)

end = mid - 1;

end while

return mid;

end leftSide

rightSide(A[],x)

start = 0; end = A.length - 1; mid = 0;

while(start <= end)

mid = (end + start)/2;

if((mid == A.length - 1 && A[mid] == x) || (A[mid] == x && A[mid + 1] > x))

return mid;

else if(A[mid] <= x)

start = mid + 1;

else if(A[mid] > x)

end = mid - 1;

end else

end while

return mid;

end rightSide

return mid;

Running Time: Binary search completes in logn time and the other primitive statements are constants so that means this algorithms completes in O(logn)

Section 1 Question 2:

English: To find the heaviest coin you must divide the coins in half making two piles and weigh them. The heavier pile must contain the heavier pile so you discard the lighter pile and repeat this process with the heavier pile until the coin is found.

Running time: In this example we divide the search space by 2 each time we check for the coin, meaning at k checks the search space is n/(2^k). At k checks the search space is equal to 1 so 1 = n/(2^k)

2^k = n

K \* Log2 = logn

K = logn

Therefore, the running time is O(logn)

Section 1 Question 3:

The running time for a binary search on a linked list is O(n). This is because we do not know the location of out middle and must loop through the entire list to find it, only then can we do binary search on it. Looping through takes n time and binary search takes logn time making the time complexity n + logn but since we only look at the highest order the answer is O(n).

Section 1 Question 4:

English: To find the square root of a number, we will do a binary search on the numbers between 0 and the number until we find one that when multiplied by itself is equal to the number

Pseudocode:

i = x/2; result = i\*i;

while(result != x)

if(result > x)

i = i/2; result = i\*I;

else if(result < x)

i = i + i/2; result = i\*I;

end while

return I;

Running Time: Binary search completes in O(logn)