Tom Fenyak

Homework Assignment 04

Exercise 1 (50 points) Binary Search

Consider the algorithm binarySearch(A, p, r, x): the description of this algorithm is provided below.

Inputs:

* a sorted array *A*
* index *p* of the first element
* index *r* of last element
* an element *x* to search

Output:

* index of element *x* in Sequence *A* if x exists in *A*
* -1 if *x* does not exist in Sequence *A*.

Algorithm description

int binarySearch(A, p, r, x)

        if (r >= l)

            midpoint = (p + r)/2;

             if A[midpoint] == x

               return midpoint;

            if A[midpoint] > x

               return binarySearch(A, p, midpoint-1, x);

else

             return binarySearch(A, midpoint+1, r, x);

        return -1;

The objective of this exercise is to derive the time complexity (running) time of the binary search. (If interested, you could read about binary search on Wikipedia)

1. (6 points) Let A = (2, 5, 9, 13, 18, 27, 34, 35, 39, 56, 63, 68, 71, 101). Assume that the index of the first element is 1.
   1. Execute manually binarySearch(A,1, A.length, 63). What is the output?

11

* 1. Execute manually binarySearch(A,1, A.length, 27). What is the output?

6

* 1. Execute manually binarySearch(A,1, A.length, 2). What is the output?

1

* 1. Execute manually binarySearch(A,1, A.length, 101). What is the output?

14

1. (2 points) Which operation should you count to determine the running time T(n) of the binary search in a sequence A of length n?

We should count the number of comparisons.

1. (12 points) Let us count the comparisons ((if A[midpoint] == x) and (if A[midpoint] > x)). Express the running time T(n) as a recurrence relation.

The number of comparisons that we would do if A[midpoint] == x would be 1 because we only have to compare x to one value in the list.

If A[midpoint] > x then we would use the if statement and search the bottom half of the list starting at 1 and going up to midpoint-1. We would be doing 2 comparisons in that case.

The recurrence relation would be T(n) = T(n/2) + 3 because we in the worst case scenario we are doing a maximum of 3 comparisons.

1. (12 points) Solve the recurrence relation T(n) using the recursion-tree method

Recursive call Tree Sum

T(n) 3 3

T(n/2) 3/2 3/2 3

T(n/4) ¾ ¾ ¾ ¾ 3

T(n/2^(i-1)) 3/2^i-1 …………………..3/2^i-1 3

Mth call (T(n/2^m = 1) T(1) …………………T(1) 3

The depth of the tree is right around log2n. When we get the sum of all the levels of the tree we come up with T(n) 3log2n. We can say that T(n) = (theta)(log2n)

1. (8 points) Solve the recurrence relation T(n) using the substitution method

The guess that I will pull for the substitution method will come from what we solved in the recursion-tree method. We will guess that T(n) = 3log2n.

First we need to find what T(n/2) is.

T(n/2) = 3log2(n/2)

= 3(log.n – log(2))

= 3(log.n – 1) since log.n is base 2.

Next we need to plug it into the original recurrence relation.

T(n) = (3(log.n – 1)) + 3

= 3log.n – 3 + 3 This proves that our guess is good!

1. (10 points) Solve the recurrence relation T(n) using the master method

For the master method we first need to find out the values of a, b, and f(n)

a = 1 (a >=1) this checks out

b = 2 (b > 1) this checks out

f(n) = 3

Next we need to compute n^logb.a

Logb.a = log2.1 = 0

With this we know that n^0 = 1

Since 1 = f(n) = (theta)(n^logb.a) Case 2 will apply here

Therefore T(n) = (theta)(n^logb.a\*lgn) = (theta)(lg.n)

Exercise 2 (20 points)

Consider the recurrence relation:

1. Solve the recurrence relation T(n) using the recursion-tree method

Recursive call Tree Sum

T(n) c c

T(n/2) c/2 c/2 c

T(n/4) c/4 c/4 c/4 c/4 c

T(n/2^i-1) c/2^i-1…………..c/2^i-1 c

T(n/2^m = 1) T(1) T(1)………………..T(1) T(1) c

We know that n/2^m = 1

Therefore n = 2^m

If m = log2(n)

The total cost is c.log2(n)

**(Theta)(log2(n)**

1. Solve the recurrence relation T(n) using the substitution method

We are going to use our total cost in the last problem as our guess.

T(n) = c.log2(n)

Using that guess we next next need to find out what T(n/2) is

T(n/2) = c.log2(n/2)

= c(log.n – log2)

= c(log.n – 1) since log.n is log base 2.

Next we need to plug in T(n/2) into our original recurrence relation

T(n) = c.(logn – 1) + c

= c.logn – c + c

= c.logn This proves our guess is correct!

1. Solve the recurrence relation T(n) using the master method

For the master method we first need to find out the values of a, b, and f(n)

a = 1 (a >=1) this checks out

b = 2 (b > 1) this checks out

f(n) = c

Next we need to compute n^logb.a

Logb.a = log2.1 = 0

With this we know that n^0 = 1

Since 1 = f(n) = (theta)(n^logb.a) Case 2 will apply here because c is some constant

Therefore T(n) = (theta)(n^logb.a\*lgn) = (theta)(lg.n)

Exercise 3 (5 points)

Consider the recurrence relation . Solve this recurrence relation using the master method (if it applies).

For the master method we first need to find out the values of a, b, and f(n)

a = 2 (a >=1) this checks out

b = 4 (b > 1) this checks out

f(n) = sqrt(n)

Next we need to compute n^logb.a

Logb.a = log4.2

Logb.a = 4^x = 2 = .5

Then logb.a = n^.5 or sqrt.n

This proves Case 2 because f(n) = (theta)(n^.5) then T(n) = (Theta)(n^logb.a\*lgn)

Exercise 4 (10 points)

Consider the recurrence relation T(n) = 2T(n/2) + n.log(n). Solve this recurrence relation using the master method (if it applies).

For the master method we first need to find out the values of a, b, and f(n)

a = 2 (a >=1) this checks out

b = 2 (b > 1) this checks out

f(n) = sqrt(n)

Next we need to compute n^logb.a

Logb.a = log2.2 = 1

This proves Case 3 because f(n) = (theta)(1) for some constant e > 0 and if a.f(n/b) <=c.f(n) for some constant c < 1. We end up getting 1<= 1 which checks out!

Exercise 5 (5 points)

Consider the recurrence relation T(n) = 0.5T(n/2) + 1/n. Solve this recurrence relation using the master method (if it applies).

For the master method we first need to find out the values of a, b, and f(n)

a = .5 (a >=1) this DOES NOT check out

b = 2 (b > 1) this checks out

f(n) = 1/n

Exercise 5 (10 points)

Consider the recurrence relation T(n) = 3T(n/2) + n. Solve this recurrence relation using the master method (if it applies).

For the master method we first need to find out the values of a, b, and f(n)

a = 3 (a >=1) this checks out

b = 2 (b > 1) this checks out

f(n) = n

Next we need to compute n^logb.a

Logb.a = log2.3 = 1.585

Case 1 would apply in this instance because we can have this prove true as long as e is .585 > e > 0. And we can say that T(n) = (theta)(n^1.855)