**Chapter 10**

***R-10.6*** Give an example set of 8 characters and their associated frequencies so that the Huffman tree for this set is a complete binary tree.

**Answer:** To design a Huffman tree for a set of 8 characters for a complete binary tree let’s consider the word “ACQUIRED” which after sorted comes ‘A’, ‘D’, ‘D’, ‘E’, ‘I’, ‘Q’, ‘R’, ‘U’

First, we remove the two minimum elements from the priority queue: A, U

2

A U

We reinsert the new root node in the priority queue, then we remove two minimum elements from the priority queue again: C, Q

2

C Q

We reinsert the new root node in the priority queue, then we remove two minimum elements from the priority queue again: D, R

2

D R

We reinsert the new root node in the priority queue, then we remove two minimum elements from the priority queue again: E, I

2

E I

We reinsert the new root node in the priority queue, then we remove two minimum elements from the priority queue again: DR, EI

4

2 2

E I D R

We reinsert the new root node in the priority queue, then we remove two minimum elements from the priority queue again: AU, CQ

4

2 2

A U C Q

We reinsert the new root node in the priority queue, then we remove two minimum elements from the priority queue again: AUCQ, DREI

8

4 4

2 2 2 2

A U C Q D R E I

|  |  |  |
| --- | --- | --- |
| Character | Frequency | HUFFMAN CODE |
| A | 1 | 000 |
| C | 1 | 010 |
| Q | 1 | 011 |
| U | 1 | 001 |
| I | 1 | 111 |
| R | 1 | 101 |
| E | 1 | 110 |
| D | 1 | 100 |

The left side is 0 and the right hand is 1at every level.

The time Complexity is: O (n log n)

***C-10.5*** Describe an efficient greedy algorithm for making change for a specified value using a minimum number of coins, assuming there are four denominations of coins (called quarters, dimes, nickels, and pennies), with values 25, 10, 5, and 1, respectively. Argue why your algorithm is correct.

**Answer:** A greedy algorithm is an algorithm that follows problem solving heuristic of making the most optimal choice. The reason greedy algorithm is opted is because it either gives maximum or minimum. In minimum number of coins best way is to get the coins with the greatest value until it smaller coins are a requirement to reach the number.

Suppose N= 52

We can opt various methods:

1. 2 \* 25 and 2 \*1 total coins = 3
2. 5 \* 10 and 2 \*1 total coins = 7
3. 10 \* 5 and 2 \*1 total coins = 12

Now to get the minimum number of coins number 1 is the most optimal solution when greedy algorithm is applied, it implements the that method. Algorithm would take O(n) time and sorting the coin would give total of O (n log n).

***A-10.1*** In the *art gallery guarding* problem we are given a line *L* that represents a long hallway in an art gallery. We are also given a set *X* = *{x*0*, x*1*, . . ., xn−*1*}* of real numbers that specify the positions of paintings in this hallway. Suppose that a single guard can protect all the paintings within distance at most 1 of his or her position (on both sides). Design an algorithm for finding a placement of guards that uses the minimum number of guards to guard all the paintings with positions in *X*.

**Answer:** Given: all the art work is in a single line L Set X {x0, x1, x2, …., xn-1} where x0, x1, x2, …., xn-1 is the position of each painting.

Now, to find the minimum number of guards for the painting greedy algorithm is the best algorithm which would aim to place each guard within distance of at most 1 on both sides.

To properly align all the paintings quick sort or merge sort is applied to sort all the paintings and now at X0 we do not need to keep a guard at the 1st painting as it is not mentioned that it is a requirement so we keep the guards in the paintings.

The input is the position of the paintings that is set X and the output would be the minimum number of guards which would guard the paintings.

Let Guard\_optimal=0;

While (X! =0) { remove paintings Xi such that its distance is smallest, X0

if(Guard(i) does not have any conflict with painting Xi) then schedule Xi for G(i)

else for the current painting X[i+2] add a new guard

Guard\_optimal = Guard\_optimal+2

Schedule painting Xi for guard Guard\_Optimal

}

It takes O(n) time. And if sorting is required it would take O (n log n)

**Chapter 11**

***R-11.1*** Characterize each of the following recurrence equations using the master theorem (assuming that *T*(*n*) = *c* for *n < d*, for constants *c >* 0 and *d ≥* 1).

a. *T*(*n*) = 2*T*(*n/*2) + log *n*

b. *T*(*n*) = 8*T*(*n/*2) + *n*2

c. *T*(*n*) = 16*T*(*n/*2) + (*n* log *n*)4

d. *T*(*n*) = 7*T*(*n/*3) + *n*

e. *T*(*n*) = 9*T*(*n/*3) + *n*3 log *n*

**Answer:** a. *T*(*n*) = 2T(n/2) + log n

Here, 𝑎, b=2

f(n) = log n by case1,

f(n) = O (nlogba- ɛ)

f(n) = O (nlog22- ɛ)

f(n) = O (n1- ɛ)

We get same log n is O (n1- ɛ), by case 1 in masters’ theorem,

T(n) = 𝜃 (nlogba)

T(n) = 𝜃 (n)

b. T(n) = 8T(n/2) + n2

a=8, b=2 and f(n) = n2

In this case, 𝑛logba = 𝑛log28=n3,

Here f(n) is O (n3- ɛ) for ɛ=1

T(n)= 𝜃(n3) by master theorem

c. T(n) = 16T(n/2) +(n log n)4

a=16, b=2 and f(n) = (n log n)4

In this case, 𝑛logba = 𝑛log216=n4,

f(n) = 𝜃 (𝑛logba logkn) where k=1

By Case 2 in theorem, T(n) = 𝜃 (𝑛logba logk+1n)

This means by case 2 in Theorem,

T(n) = 𝜃 (n4(log n)5)

d. T(n) = 7T(n/3) + n

a = 7, b=3 and f(n) =n

𝑛logba=nlog37=n1.77

By Case 1 of Theorem, f(n) = n is O (n1.77) where 𝜀=0.77

Hence, T(n) = 𝜃 (𝑛logba)

T(n) = 𝜃(n1.77)

e. T(n) =9T(n/3) +n3log n

a=9, b=3

f(n) = n3 log n

In this case, 𝑛logba=nlog39=n2

By vase 3 of the theorem, f(n) = n3log n is Ω(n2) where 𝜀= 1

→a f(n/b) ≤𝛿 f(n)

LHS: a f(n/b)

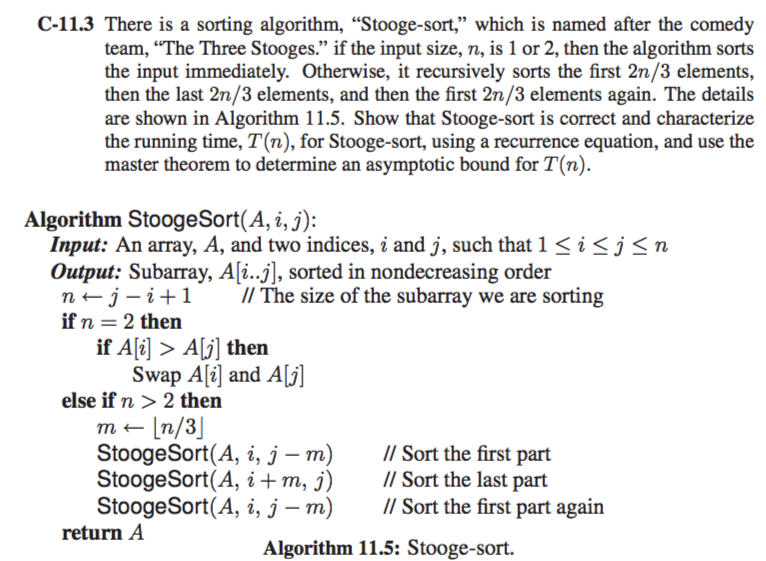
→9 f(n/3) →9(n/3)3 log n/3 →n3/3log n/3

So, n3/3 log n/3≤ 𝛿𝑛3 log n for 𝛿=1/3 𝑎𝑛𝑑 𝑛≥1

Hence, T(n) is 𝜃 (n3 log n)

***C-11.3*** There is a sorting algorithm, “Stooge-sort,” which is named after the comedy team, “The Three Stooges.” if the input size, *n*, is 1 or 2, then the algorithm sorts the input immediately. Otherwise, it recursively sorts the first 2*n/*3 elements, then the last 2*n/*3 elements, and then the first 2*n/*3 elements again. The details are shown in Algorithm 11.5. Show that Stooge-sort is correct and characterize the running time, *T*(*n*), for Stooge-sort, using a recurrence equation, and use the master theorem to determine an asymptotic bound for *T*(*n*).

**Answer:**



The algorithm stooge sort works by simple rule divide and conquer.

Let’s consider following example= (6,3,1,2,8,9)

Stooge start will start by first sorting 2n/3 elements of the series.

So, it will sort the first 4 numbers- {6,3,1,2} stooge sort will give 6,31 and 2. And then further dividing 6,3 and 1.

Since two number are there in the set they get swapped if A[i] >A[i+1]

Since the two are sorted the merge starts and numbers are added (1,2,3,6,)

Here, the first set was sorting all the large numbers were pushed towards end and when 2nd sorting started in both set the first 4 numbers are common in both set and the final set would be 1,2,3,6,8,9

Assume stooge sort sorts correctly sorts an input array A[1to k] where k is the length of [A] and 1≤k <n.

By induction hypothesis first time stooge is called sort (a, i, j-k) it correctly sorts 2n/3 eleements, so that elements 1 to n/3 are less than elements (n+1)/3 to 2n/3

Then it sorts last 2n/3 elements so that the elements (n+1)/3 are less than elements 2(n+1)/3 to n which are largest n/3 elements in a. the last call to stooge-sort (a, I, j-k) sorts correctly. The sorted elements are less than elements 2(n+1)/3.

Time complexity T(n)= o(𝑛log3/23)

***A-11.1*** A very common problem in computer graphics is to approximate a complex shape with a *bounding box*. For a set, *S*, of *n* points in 2-dimensional space, the idea is to find the smallest rectangle, *R*, with sides parallel to the coordinate axes that contains all the points in *S*. Once *S* is approximated by such a bounding box, we can often speed up lots of computations that involve *S*. For example, if *R* is completely obscured some object in the foreground, then we don’t need to render any of *S*. Likewise, if we shoot a virtual ray and it completely misses *R*, then it is guarantee to completely miss *S*. So, doing comparisons with *R* instead of *S* can often save time. But this savings is wasted if we spend a lot of time constructing *R*; hence, it would be ideal to have a fast way of computing a bounding box, *R*, for a set, *S*, of *n* points in the plane. Note that the construction of *R* can be reduced to two instances of the problem of simultaneously finding the minimum and the maximum in a set of *n* numbers; namely, we need only do this for the *x* coordinates in *S* and then for the *y*-coordinates in *S*. Therefore, design a divideand- conquer algorithm for finding both the minimum and the maximum element of *n* numbers using no more than 3*n/*2 comparisons.

**Answer:** There is an array A that has n numbers. Maximum and minimum from the array is supposed to be deduced, divide and conquer method can get this result.

Let’s name the algorithm MM. Here, the input would be array containing n numbers in it and output would be Minimum and Maximum number from array. The array is divided into 2 equal arrays a1 and a2

Here (mi1, ma1) = MM(a1)

(mi2, ma2) = MM(a2)

if (mi1 > mi2) then return minimum = mi2

else return minimum = mi1

if (ma1 > ma2) then return maximum = ma1

else return maximum = ma2

The time complexity of T(n) = 2T(n/2) + O (1) by masters’ theorem

a, b = 2 and f(n) = O (1),

𝑛logba=nlog22=n

Here, f (n) is O (n).

So, by case 1 in masters’ theorem T(n)=O(𝑛log22) = O (n)

T(n) can be performed 3n/2 comparisons, since T(n) = O (n)

**Chapter 12**

***R-12.9*** Sally is hosting an Internet auction to sell *n* widgets. She receives *m* bids, each of the form “I want *ki* widgets for *di* dollars,” for *i* = 1*,* 2*, . . ., m.* Characterize her optimization problem as a knapsack problem. Under what conditions is this a 0-1 versus fractional problem?

**Answer:**

If all the widgets are sold or none of them are sold then it would be considered as a 0-1 knapsack problem. If only some of the widgets are sold then it would amount to a fractional knapsack problem, as it allows fractional quantitates.

***A-12.3*** An American spy is deep undercover in the hostile country of Phonemia. In order not to waste scarce resources, any time he wants to send a message back home, he removes all the punctuation from his message and converts all the letters to uppercase. So, for example, to send the message, “Abort the plan! Meet at the Dark Cabin.” he would transmit

ABORTTHEPLANMEETATTHEDARKCABIN

Given such a string, *S*, of *n* uppercase letters, describe an efficient way of breaking it into a sequence of valid English words. You may assume that you have a function, valid(*s*), which can take a character string, *s*, and return true if and only if *s* is a valid English word. What is the running time of your algorithm, assuming each call to the function, valid, runs in *O*(1) time?

**Answer:**

Input of the algorithm is a string that is supposed to be converted and the output is string consisting a valid English word of S.

Let O be an empty list here

for i←0 to n-q do

if valid (Substring (0, i)) then

O.add (Substring (0, i))

O.add (StringBreaker(Substring(i, n-1)))

break

return O

It normally takes O (n) time to check Substring so the total running time is O (n2)