# Algorithm Design Assignment 1

## Chapter 1 - Exercise 14

### There are two algorithms called Alg1 and Alg2 for a problem of size *n*. Alg1 runs in *n*2 microseconds and Alg2 runs in 100*n* log *n* microseconds. Alg1 can be implemented using 4 hours of programmer time and needs 2 minutes of CPU time. On the other hand, Alg2 requires 15 hours of programmer time and 6 minutes of CPU time. If programmers are paid 20 dollars per hour and CPU time costs 50 dollars per minute, how many times must a problem instance of size 500 be solved using Alg2 in order to justify its development cost?

The cost of implementing Alg1 is:

* Programmer time cost: 4 hours \* $20/hour = $80
* CPU time cost: 2 minutes \* $50/minute = $100
* Total cost of Alg1: $80 + $100 = $180

The cost of implementing Alg2 is:

* Programmer time cost: 15 hours \* $20/hour = $300
* CPU time cost: 6 minutes \* $50/minute = $300
* Total cost of Alg2: $300 + $300 = $600

The running time of Alg1 for a problem instance of size 500 is:

* n^2 = 500^2 = 250,000 microseconds

The running time of Alg2 for a problem instance of size 500 is:

* 100n \* log n = 100 \* 500 \* log(500) ≈ 100 \* 500 \* 2.7 = 135,000 microseconds

The cost of running Alg1 for a problem instance of size 500 is:

* (250,000 microseconds / 1,000,000) \* 2 minutes \* $50/minute = $25

The cost of running Alg2 for a problem instance of size 500 is:

* (135,000 microseconds / 1,000,000) \* 6 minutes \* $50/minute = $40.5

Number of problem instances = Implementation cost difference / Running cost difference per problem instance

* Number of problem instances = $420 / $15.5 ≈ 27.1

a problem instance of size 500 must be solved 28 times using Alg2 in order to justify its development cost.

## Chapter 1 - Exercise 33

### Give an algorithm for the following problem and determine its time complexity. Given a list of *n* distinct positive integers, partition the list into two sublists, each of size *n/*2, such that the difference between the sums of the integers in the two sublists is maximized. You may assume that *n* is a multiple of 2.

Algorithm:

1. Sort the list in ascending order.
2. Create two empty sublists.
3. Add the first *n*/2 elements to the first sublist.
4. Add the remaining elements to the second sublist.
5. Return the two sublists.

Program in java

A screenshot of a computer program

Description automatically generated

Time complexity:

1. Sorting the array in ascending order takes = O(n logn)
2. it iterates over the array twice, once to add the first n/2 elements to a sublist and once to add the remaining elements to another sublist. This takes O(n) time the program calculates the sum of each sublist and prints the difference between the two sums. This also takes O(n) time.
3. Therefore, the overall time complexity of the program is O(n log n) + O(n) + O(n) = O(n log n).

## Chapter 2 - Exercise 4

### Show that the worst-case time complexity for Binary Search (Algorithm 2.1) is given by when n is not restricted to being a power of 2. Hint: First show that the recurrence equation for W(n) is given by

### 

### To do this, consider even and odd values of n separately. Then use induction to solve the recurrence equation.

To show that the worst-case time complexity for Binary Search is given by *O*(log*n*) when *n* is not restricted to being a power of 2

Show that the recurrence equation for *W*(*n*) is given by: W(n) = 1 + W(n/2)

For even values of *n*, this is straightforward, since the binary search algorithm will divide the array in half and then recurse on one of the two halves.

For odd values of *n*, the binary search algorithm will first round *n* down to the nearest even number and then recurse on the resulting array.

This means that the worst-case time complexity for binary search on an odd-sized array is the same as the worst-case time complexity for binary search on an even-sized array.

Use induction to solve the recurrence equation:

The base case is *n*=1, for which the worst-case time complexity is clearly 1.

For the inductive step, we assume that the recurrence equation holds for all values of *n* less than *k*, and we show that it also holds for *n*=*k*.

W(k) = 1 + W(k/2)

By the inductive hypothesis, we know that *W*(*k*/2)=log(*k*/2). Therefore, we can substitute this into the recurrence equation to get:

W(k) = 1 + log(k/2)

W(k) = log(2k/2)

W(K) = log(k)

Therefore, we have shown that the recurrence equation holds for *n*=*k*, which completes the inductive step.

The recurrence equation for the worst-case time complexity of binary search using induction is *W*(*n*)=1+*W*(*n*/2). The solution to the recurrence equation is *W*(*n*)=log(*n*). Therefore, the worst-case time complexity for binary search is *O*(log*n*).

## Chapter 2 - Exercise 16

### Suppose that, in a divide-and-conquer algorithm, we always divide an instance of size n of a problem into 10 subinstances of size n/3, and the dividing and combining steps take a time in Θ(n2) . Write a recurrence equation for the running time T(n), and solve the equation for T(n).

The recurrence equation for divide and conquer algorithm is given by Master theorem formula.

F(n) = af(n/b) + g(n)

Where,

* f(n) is the number of operations needed to solve a problem.
* n is the size of the problem.
* a is the number of subproblems in the recursion.
* n/b is the size of each subproblem
* g(n) is computations needed to combine the solutions from subproblems into overall solution to the original problem.

In the given problem, f(n) = T(n), n = n, a = 10 , b = 3 and g(n) = Θ(n2).Using the above, recurrence equation for the running time T(n) is

T(n) = 10f(n/3) + Θ(n2)

The master theorem states that the running time of a divide-and-conquer algorithm is given by one of the following three cases:

* Case 1: If *a*<*bd*, then *T*(*n*)=*O*(*f*(*n*)).
* Case 2: If *a*=*bd*, then *T*(*n*)=*O*(*f*(*n*)log*n*).
* Case 3: If *a*>*bd* and *f*(*n*)=*O*(*nϵ*) for some *ϵ*<*d*, then *T*(*n*)=*O*(*f*(*n*)log*n*).

In this case, we have *a*=10, *b*=3, and *d*=log3​10=3.3219. We also have *f*(*n*)=*O*(*n*2). Since *a*>*bd* and *f*(*n*)=*O*(*nϵ*) for some *ϵ*<*d*, we are in Case 3 of the master theorem. Therefore, the running time of the algorithm is given by:

T(n) = O(f(n) log n) = O(n^2 log n)

In conclusion, the solution to the recurrence equation for the running time of the divide-and-conquer algorithm you described is *T*(*n*)=*O*(*n^*2log*n*).