**CS375 Assignment 1**

(Due on September 15 by 11:59pm)

The assignment is designed to enhance the concept of time complexity, brute-force algorithm design, instruction counts, mathematics computation, and to implement an efficient program to solve a practical problem, proof of correctness.

There are two parts in this assignment: (A) Theory part and (B) programming part

**[Part A] Theory [85%]:**

1. [8%] Design an algorithm: Given a list of n distinct positive integers (you may assume that n is a multiple of 2),
2. 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; show the time complexity;
3. 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 minimized; show the time complexity;

**Answer:**

1. First, sort the list of n distinct positive integers in descending order so that the first number is the greatest, with n/2 being in the middle. Then, initialize two sublists, we can call them sublist1 and sublist2. Add all the elements that are greater than n/2 into sublist1 and add all the elements that are less than n/2 into sublist2. Since n/2 is in the middle of the sorted list, both sublists will have a size of n/2. The difference between the sums of the integers in these two sublists is maximized because sublist1’s sum is the largest possible in the size of n/2 and sublist2’s sum is the smallest possible in the size of n/2. The time complexity is O(nlogn).
2. First, calculate the total sum (total\_sum) of the integers in the list. Then, initialize a 2D array called arr of size (n/2 + 1)\*(total\_sum + 1), where arr[i][j] is true if it is possible to obtain a sum of j using the first i integers, and arr[0][0] is true in case we get a sum of 0. Iterate through the n distinct positive integers in the list, where for each i from 1 to n/2, for each j from 0 to the total sum, set arr[i][j] to true if either arr[i][j] or arr[i – 1][j – arr[i – 1]] is true. To find the sum of one of the sublists, find the largest j such that arr[n][j] is true. The sum of the other sublist will be total\_sum – j. The time complexity is O((n/2)\*total\_sum).
3. [9%] Consider the following algorithm (assume n >1) :

int any-equal (int n, int A[ ][ ])

{

Index I, j, k, m;

for (i=1; i<=n; i++)

for (j=1; j<=n; j++)

for (k=1; k<=n; k++)

for (m=1; m<=n; m++)

if (A[i][j] == A[k][m] && !(i==k && j==m))

return 1;

return 0;

}

1. Show the best case; what is the best case time complexity;

**Answer:** The best case is when the algorithm finds a matching pair of elements in the first iteration of the loops, returning 1 without having to go through more comparisons.

The best case time complexity is O(1) because the algorithm only makes one comparison before returning 1.

1. Show the worst case; what is the worst case time complexity;

**Answer:** The worst case is when the algorithm finds a matching pair of elements in the 2d array A at the very last comparison, meaning that it iterated through all the possible pairs of elements before finding the matching pair. The worst case time complexity is O(n^4) because there are four nested loops that each iterate from 1 to n, so the number of comparisons at the end of the very last possible comparison is proportional to n^4.

1. Try to improve the efficiency of the algorithm;

**Answer:** Using a set data structure improves the efficiency of the algorithm:

Int any-equal (int n, int A[][])

{

Set<int> elements; // Set to store elements that were encountered

for(i = 1; i <= n; i++) {

for(j = 1; j <= n; j++) {

if(elements.encountered(A[i][j])) {

return 1; // If the set already contains the element, return 1 (algorithm done)

}

elements.add(A[i][j]); // If not in the set already, add the element to the set

}

}

return 0;

}

This algorithm has improved efficiency since it is reduced to two loops, using a set data structure to keep track of encountered elements. It iterates through the array once, storing encountered elements in the set, and returns 1 as soon as it encounters a matching pair. The time complexity of the improved algorithm is O(n^2), as opposed to it’s time complexity of O(n^4) in the previous version.

3. (8%) (Brute-force algorithm)

Alternating disks: You have a row of 2n disks of two colors, n dark and n light. They alternate: dark, light, dark, light, and so on. You want to get all the dark disks to the right-hand end, and all the light disks to the left-hand end. The only moves you are allowed to make are those which interchange the positions of two neighboring disks.



Design a brute-force algorithm for solving this puzzle and determine the number of moves it makes. (Your algorithm should not be worse than O(n2). You may describe your approach with pseudo-code.)

**Answer:**

alternatingDisksAlgorithm(array)

n = length(array) / 2 // n dark disks and n light disks

count = 0 // used to determine the total number of moves made

while True

swapped = false // tracks if any swaps were made in the iteration

for i=0 to 2n – 2 do // iterate through the disks

if array[i] == “dark” and array[i + 1] == “light” then // check if there is a dark disk to the left of a light disk

temp = array[i] // set a temp variable to the dark disk

array[i] = array[i + 1] // set the dark disk to the light disk

array[i + 1] = temp // set the light disk to the temp (dark disk), finishing the necessary swap)

count += 1 // increment the number of moves made

swapped = true // set swapped to true to continue the loop

if not swapped then break // end the loop if no swaps were made (puzzle is solved)

return count // return the total number of moves made to solve the puzzle

To determine the number of moves the algorithm will make, we follow the formula . For example, in the picture above, we have 2n = 8 disks, so n = 4. Then, it will take = = = 10 total moves to solve the puzzle.

1. [8%] Given a list of distinct numbers and the average or mean of those numbers, following pseudo-code is to determine whether there are more numbers above the average than below.

MoreAbove( list, average, N )

countAbove = 0

for j = 1 to N do

if list[ j ] > average then

countAbove = countAbove + 1

if countAbove > N/2 then return true

return false

Let’s take the “>” as the barometer operation. What is the count for the best case, and what is the count for the worst case? Give your explanation.

**Answer:** The count for the best case is N/2. This is the case if the list is sorted from greatest to least and the average is in the middle, where the algorithm stops as soon as it encounters N/2 elements that are above the average than below.  
The count for the worst case is N. This is the case if the list is sorted such that all the numbers are below average. This causes the algorithm to compare the average with every N element in the list to check if any of them are above average.

1. [9%] What is the count for the instruction CountMe as a function of n for the fragments below?
   1. [3%]  
       Line 1: **for** (i=0; i< n2+1; i++ ){

Line 2:    **for** (j=0; j<= n3/5; j++){

Line 3:       CountMe }}

**Answer:** count = (n2 + 1)\*( n3/5)

* 1. [3%]

Line 1: j = n2

Line 2: **while** (j > 0) {

Line 3:    CountMe

Line 4:    j = j-2; }

**Answer:** count = n2/2

* 1. [3%] For simplicity you may assume that n=2k for some positive integer k.

Line 1: j = 1

Line 2: **while** (j<=n2) {

Line 3:    CountMe

Line 4:    j = 2\*j;   
        }

**Answer:** count =

1. [6%] Show by using limits that:
   1. [3%]

**Answer:**

0 <= 4^(2n + 1) <= c16^n

0 <= 4^(2n + 1) / 4^(2n) <= c16^n / 4^(2n)

0 <= 4 <= c, so c = 4.

Choose N = 1, then 0 <= 4^(2\*1 + 1) <= 4\*16^1

-> 0 <= 64 <= 64. So, c > 0 and N > 0 such that 0 <= 4^(2n + 1) < c16^n for all n >= N. Therefore, .

0 <= c16^n <= 4^(2n+1)

0 <= c16^n / 4^(2n) <= 4^(2n+1) / 4^(2n)

0 <= c <= 4, so c = 4.

Choose N = 1, then 0 <= 4\*16^1 <= 4^(2\*1 + 1)

-> 0 <= 64 <= 64. So, c > 0 and N > 0 such that 0 <= c16^n <= 4^(2n + 1) for all n >= N. Therefore, .

Since and , we can conclude that .

* 1. [3%]

**Answer:**

(n^5 – n^2 + 2n) / n > 0?

= = = .

This limit indicates that f(n) = n^5 – n^2 + 2n > g(n) = n, as f(n) grows faster than g(n) to get the limit equal to infinity, so .

7. [5%] Using the definitions of O and  , show that

but

**Answer:**

Definition of : 0 <= g(n) <= cf(n) for all n >= N

0 <= 10n + 8 <= cn^2

0 <= 10 + 8/n <= cn

Choose N = 1, then 10 + 8/n000000000 <= 10 + 8/1 = 18. So, any c >= 18.

Choose c = 18, then 10 + 8/n <= 18n. So, c > 0 and N > 0 such that 0 <= 10n + 8 <= cn^2 for all n >= N. So, .

Definition of : 0 <= cf(n) <= g(n) for all n >= N

Assume that . From the definition of , there must exist c > 0 and N > 0 such that 0 <= cn^2 <= 10n + 8.

Divide both sides by n^2, 0 <= c <= 10/n + 8/n^2.

Take the limit of 10/n + 8/n^2 as n approaches infinity, which equals to 0. So, there exists an N such that c > 10/n + 8/n^2. Therefore, by proof of contradiction, .

1. (7%) Suppose f + g are two functions (taking nonnegative values) such that g = O(f). Prove f + g = θ(f); in other words, f is an asymptotically tight bound for the combined function f + g.

**Answer:**

Since g = O(f), g(n) <= cf(n) for all n >= N.

(f + g)(n) = f(n) + g(n) <= f(n) + cf(n) = (1 + c)f(n).

Choose c = 1 and n = N, then f(N) + g(N) <= 2f(N). So, f + g = O(f).

Since g = O(f), g(n) <= cf(n) for all n >= N.

(f + g)(n) = f(n) + g(n) >= f(n) - cf(n) = (1-c)f(n).

Choose c = 1 – c and n = N, then f(N) + g(N) >= -cf(n). So, f + g = (f).

Since f + g = O(f) and f + g = (f), f + g = θ(f).

9. [7%] Given the following piece of code, show the loop invariant and prove its correctness based on the loop invariant. (assume X[i] are known).

Line 1: A[0]=X[0]; s=X[0]\*X[0];   
Line 2: for i=1 to n-1 do   
Line 3:     s= s + X[i]\* X[i]   
Line 4:     A[i] = sqrt(s)  
Line 5: return array A

**Answer:**

The loop invariant is that at the start of each iteration of the loop, the variable s is the sum of the squares of every element in the array X from X[0] to X[i – 1] and A[i – 1] equals to the square root of s.

s = X[0]^2 + X[1]^2 + . . . + X[i – 1]^2

A[i – 1] = sqrt(s)

Initialization:

Before the for loop starts, s is equal to X[0]^2 and A[0] is equal to sqrt(s). So, the loop invariant is satisfied for the start of the loop.

Maintenance:

Assume that i = k for the loop invariant. Then, in the kth iteration of the loop, s = s + X[k]\*X[k] and A[k] = sqrt(s). Therefore, at the start of the (k+1)th iteration, s = X[0]^2 + X[1]^2 + . . . + X[k]^2 and A[k] = sqrt(s). So, the loop invariant is true for i = k and i = k + 1.

Termination:

At the end of the for loop, s = X[0]^2 + X[1]^2 + . . . + X[n – 1]^2 and A[n – 1] = sqrt(s). So s = X[0]^2 + X[1]^2 + . . . + X[i -1]^2 and A[i – 1] = sqrt(s). The loop invariant is correct.

1. (8%)



1. What is the time complexity of the above algorithm; (2%)

**Answer:** The time complexity of the above algorithm is O(n^3).

1. Improve the above algorithm with an asymptotically better running time. Show your algorithm, and the time complexity (3%)

**Answer:**

sum = 0

for i = 1, 2, . . ., n

sum = A[i]

for j = i + 1, i + 2, . . ., n

sum += A[j]

Store the sum in B[i,j]

endfor

endfor

The time complexity is O(n^2).

1. Proof of correctness of your algorithm (3%)

Proof by induction.

Base Case: i = 1

The first iteration of the outer loop sets i = 1. The algorithm finds the sum of integers from A[1] to A[j] for all j where j > 1. Since there is only one element in B[1][j], it successfully finds the sum of integers from A[1] to A[j].

Inductive Hypothesis:

After the ith iteration of the outer loop, the algorithm successfully finds the sum of elements from A[i] to A[j] for all j where j > i.

Inductive Step:

Assume that after the kth iteration of the outer loop, for some k >= 1, the algorithm successfully finds the sum of integers from A[k] to A[j] for all j where j > k. Now, consider the (k + 1)th iteration of the outer loop, which sets i = k + 1. During this iteration, the algorithm sets the sum to A[k + 1]. Then, it enters the inner loop, iterating through j where j > k + 1. After each iteration of the inner loop, A[j] is added to the sum. Thus, the algorithm finds the sum of integers from A[k + 1] to A[j] for all j where j > k + 1. By the inductive hypothesis, we know that after the kth iteration, the algorithm successfully finds the sum of integers from A[k] to A[j] for all j where j > k. As such, the sum starts with A[k + 1] and A[j] is added to it after each iteration of the inner loop during the (k + 1)th iteration of the outer loop. So, the algorithm is correct, successfully finding the correct values for B.

1. [10%] **Award problem to students with blind tags**

**Problem:** There are 30 students attending a university award ceremony. Each student has a tag labeled by either “A”, “B”, or “C”. The tag is stick on the back of each student, which is visible only by the others except the student self. Students are not allowed to communicate each other. In other words, each student is aware of other students’ labels but not aware of his/her own label. Suppose there are 5 students with “A”, 10 students with “B”, and 15 students with “C”. During the ceremony, the university president announces that the award is given to students who have the label “A”, and asks the students to come to the podium to receive the award certificate. (1) (7%) Explain how the students with “A” can figure out their label without communicating with other students and receive the certificate? (2) (3%) Prove your solution by induction.

**Note:** For this problem, it is assumed that students are aware that at least one student will receive the award certificate. After the announcement, the president will give maximum 30 chances to all students to figure out their own labels, with 1 minute for each chance. For example, the president will give the 1st minute to students to figure out their label, whoever figures out the own label with “A” will come to receive the certificate. If no one figures it out, the president will give a second chance (with 2nd minute), or continuing the third chance (3rd minute), fourth chance (4th minute), until thirtieth chance (30th minute) at maximum or stop at any round if all the students with “A” have figured it out and received the certificates.

**Answer:**

1. All the students can line up in any order. When the president announces the first chance, the 30th student, who is at the end of the line, checks the labels of all the other students. If they see 10 “B” labels and 15 “C” labels, then they must have a tag labelled “A”, so they will realize so and come to the podium to receive the award certificate. If they do not see 10 “B” labels or 15 “C” labels, then they must not have a tag labeled “A”. The president will then announce the second chance, and the next student (29th) will repeat the same process of checking everyone’s tags, including the student who received the certificate. This process repeats until the thirtieth chance (30th minute), which is the worst case, or if five students labelled “A” have come up and received their certificates.
2. This solution can be proved by induction.

Base Case: n = 1

When n = 1, which is the first minute, it is possible for the 30th student to find their label to be “A” if they see 10 “B” labels and 15 “C” labels among the other students.

Inductive Hypothesis:

Assume that for some k > 0, if k students labelled “A” have already received their certificates, then the (30 – k)th student can find their label to be “A” if they see 10 “B” labels and 15 “C” labels.

Inductive Step:

Consider the (30 – k – 1)th student. They can see the labels of the (k + 1) students ahead of them, including the (30 – k)th student who already received their certificate. By the inductive hypothesis, the (30 – k)th student correctly found their label to be “A”. Therefore, the (30 – k – 1)th student can also correctly find their label to be “A” if they see 10 “B” labels and 15 “C” labels among the other students. So, each student with a tag labeled “A”will correctly figure out their label and come to receive their award certificates within 30 minutes. So, the solution works in this problem where the students with “A” can figure out their label without communicating with other students and receive the certificate.

**[Part B]: Programming (15%)**

1. Given an array containing n keys, design an algorithm to determine whether there is such a key that is equal to the difference between the other two keys in the array. Explain the worst-case time complexity. (No sequential search is allowed).

Example data for test:

[14, 89, 18, 105, 23, 4, 35, 99, 67, 76, 198, 20, 5, 38, 55, 2, 30, 19, 487, 11, 40, 10, 13, 27, 22, 45, 37, 231, 46, 17, 731, 49, 167, 234, 59, 91, 179, 201]

2. Write-up:

1. Provide a program package, a readme file to explain how to run your program, and show your code and results. (9%).
2. Explain your implemented algorithm and its worst-case time complexity (3%)

**Answer:**

My implemented algorithm first opens an input file and an output file and reads the keys from the input file into an array, arr, with the number of keys read being stored in variable n. The algorithm then calculates the absolute difference between each pair of keys in the array, setting the corresponding index for each difference in a hash map (aka dictionary) to 1. It also stores the pairs that have each difference in two other arrays, pair1 and pair2, which makes sure that the larger key is stored first (in regard to 4.9b). Next, the algorithm iterates through the array again to find any key that is in the hash map. If so, then it has found a match with the key and the difference between the other two keys, printing the key and the pair whose difference that it is equal to (in decreasing order) in the output file. A Boolean variable, match, is set to true to indicate that a match has been found. If there is no match, then match stays false, which causes the program to open the output file again in write mode, clearing the file and returning a 0 byte blank file.

The worst-case time complexity is O(n^2) This is because the first two nested loops run in O(n^2) time as they iterate through the array and compare the keys with the differences.

*Extra points* (5%): Print out all the keys that are equal to the difference between the other two keys in the array, and print out the corresponding two keys.

3. Program demo (3%)

TA will run your program. You will also be requested to demo your program in the lab section. To test your program, you are required to provide your own test sets (contained in the inputFile.txt). For example, you should identify the test sets including cases with such key, without such key, or with more than one such key.

In addition, your program will be tested by a new inputFile.txt which is designed by TA as well.

4**. Part B: Submission specification**

4.1 For Part B, submit one zip file of your code package to the blackboard. The zip file includes your code, makefile, and write-up (.doc).  The title is:

firstname\_lastname\_programming\_language\_used.tar.gz or  
firstname\_lastname\_programming\_language\_used.zip

4.2 All your files including a Makefile must reside in one TOP level directory named "firstname\_lastname\_programming\_language\_used".

4.3 The TA will compile your code using your Makefile. That should result  
in the executable files. For this assignment there would be just one executable file. It should be named “submission”. This will be created by running the Makefile supplied by you.

4.4 Your program MUST take command line arguments.

4.5 Your program will read an input file and write an output file.

4.6 Your program should be invoked like this…

prompt>submission inputFile.txt outputFile.txt

where inputFile.txt is referring to an input file,

ouputFile.txt is referring to an output file.

4.7 Sample input & output files will be provided. Please adhere to the file formats specified by the examples (2023Fall\_SampleInputOutputCode\_TA.zip).

4.8 For this assignment there would be one input file, which will have one key per line. The number of lines is variable; follow the example to know how to read a variable length file.

4.9 For this assignment the output file will contain either of…

* 1. Just one key – which is the answer to the main problem. This key should be the difference of any two keys. If you don’t find such a key, then write a 0 byte blank file.
  2. Or if you are attempting the extra credit then you write all instances of a key matching the difference of two other keys. In which case each line will have 3 keys. The first key should be the difference of the other two keys. The other two keys should be in decreasing order. Write the 3 keys with whitespaces to delimit. Please look at the example to understand. If there are no such matching keys then you output a 0 byte blank file.

4.10 To test your program, you are required to provide your own test sets (contained in the inputFile.txt). For example, you should identify the test sets including cases with such key, without such key, or with more than one such key.

Your program will be tested by the inputFile.txt which is designed by TA as well.

5. One zipped electronic file (Part B) must be submitted through the Brightspace.