COP 3502- Algorithm Analysis Exercise

Part A

1) For an $O(n^3)$ algorithm, one data set with n = 3 takes 54 seconds. How long will it take for a data set with n = 5?

Solution

Let T(n) be the function for the run time of the algorithm. Then, $T(n) = cn^3$ for some constant c.
$T(3) = c3^3 = 54$
$27c = 54$, so $c = 2$
$T(5) = c5^3 = 2(125) = 250$ seconds.

2) For an $O(2^n)$ algorithm, a friend tells you that it took 17 seconds to run on her data set on a $O(2^n)$ algorithm. You run the same program, on the same machine, and your data set with n = 7 takes 68 seconds. What size was her data set?

Solution

Let T(n) be the function for the run time of the algorithm. Then, $T(n) = c2^n$ for some constant c.
$T(7) = c2^7 = 68$
$128c = 68$, so $c = 68/128 = 17/32$.
$T(n) = c2^n = 17(2^n)/32 = 17$, so $2^n = 32$ and $n = 5$.

3) For an $O(N^k)$ algorithm, where k is a positive integer, an instance of size M takes 32 seconds to run. Suppose you run an instance of size 2M and find that it takes 512 seconds to run. What is the value of k?

Solution

Let T(n) be the function for the run time of the algorithm. Then, $T(n) = cn^k$ for some constant c.
$T(m) = cm^k = 32$

$T(2m) = c(2m)^k = c2^k m^k = 512$, but since $cm^k = 32$, substituting, we have:
$32(2^k) = 512$
$2^k = 16$
$k = 4$

4) Assume that an $O(\log_2 N)$ algorithm runs for 10 milliseconds when the input size (N) is 32. What input size makes the algorithm run for 14 milliseconds?

Solution

Let T(n) be the function for the run time of the algorithm. Then, $T(n) = c\log_2 n$ for some constant c.
$T(32) = c\log_2 32 = 10$
$5c = 10$, so $c = 2$
$T(n) = 2\log_2 n = 14$, so $\log_2 n = 7$ and $n = 2^7 = 128$.

5) An algorithm to process a query on an array of size n takes O($\sqrt{n}$) time. For $n = 10^6$, the algorithm runs in 125 milliseconds. How many ***seconds*** should the algorithm take to run for an input size of n = 64,000,000?

Solution

Let the algorithm with input array size n have runtime $(n) = c\sqrt{n}$, where c is some constant.

Using the given information, we have:

$$T(10^6) = c\sqrt{10^6} = 125ms$$

$$c(1000) = 125ms$$

$$c = .125ms = \frac{1}{8}ms$$

Now, solve for the desired information:

$$T(64 \times 10^6) = c\sqrt{64 \times 10^6}$$

$$= \frac{1ms}{8} \times \sqrt{64} \times \sqrt{10^6}$$

$$= \frac{8 \times 1000ms}{8} = 1000ms = 1second$$

7) An algorithm processing a two dimensional array with R rows and C columns runs in (RC^2) time. For an array with 100 rows and 200 columns, the algorithm processes the array in 120 ms. How long would it be expected for the algorithm to take when processing an array with 200 rows and 500 columns? Please express your answer in ***seconds. // for this question,the function should look like this:*** $T(R, C) = kRC^2$ ***where k is a constant***

Solution

Let the algorithm with input size n have a runtime of $(R, C) = kRC^2$, for some constant k. Using the given information we have:

$(100,200) = k(100)(200)^2 = 120ms$

k=120ms/ 4×10^6

Now, we must find T(200,500):

$(200,500) = k(200)(500)^2 = 120\text{ms}/(4\times 10^6)\times (200)(500)^2 = 1500ms = 1.5s$

Thus, our final answer is **1.5 seconds.**

8) A search algorithm on an array of size n runs in O(lg n) time. If 200,000 searches on an array of size 2^{18} takes 20 ms, how long will 540,000 searches take on an array of size 2^{20} take, in milliseconds?

Let T(n) represent the time for a single search on an array of size n, where $T(n) = clog_2 n$. Using the given information, we have:

$$200000 \times T(2^{18}) = 200000 \times c \times log_2 2^{18} = 20ms$$
$$18 \times 200000c = 20ms$$
$$c = \frac{1}{180000} ms$$

Now, we would like to find 540000T(2^{20}):

$$540000 \times T(2^{20}) = 540000 \times \frac{1}{180000} ms \times log_2 2^{20}$$
$$= 3 \times 20ms$$
$$= 60ms$$

Part B

The following questions, represented as functions with appropriate names, determine the run-time for the function in terms of the variable n. The answers should simply be Big-Oh answers, but you need to provide ample justification for your answers. You may assume that n is a positive integer.

Question 1

```
int function5(int A[], int B[], int n) {
        int i, j, sum = 0;
        for (i=0; i<n; i++)
                for (j=0; j<n; j++)
                        if (A[i] == B[j])
                                sum++;
        return sum;
}
```

Solution

The if statement gets executed n^2 times. Namely, it get executed for each ordered pair (i,j), ranging from (0,0) to (n-1,n-1). This is the portion that dominates the code, so this function runs in $O(n^2)$ time.

Question 2

```
int function6(int A[], int B[], int n) {
        int i=0,j=0;
        while (i < n) {
                while (j < n && A[i] > B[j]) j++;
                i++;
        }
        return j;
}
```

Solution
This one's tricky! On initial observation, we might think that we have two nested loops that could each run n times. But, on closer inspection, we see that j can only be incremented n times and i can only be incremented n times and each loop iteration must have one or the other increment. Thus, at most, 2n steps can run before both loops exit. Thus, the run time is O(n).

Question 3

```
int function7(int A[], int B[], int n) {
        int i=0,j;
        while (i < n) {
                j=0;
                while (j < n && A[i] > B[j]) j++;
                i++;
        }
        return j;
}
```

Solution
The key difference here is that j is reset to 0 each time, so that inner loop really can run n times every single time. This changes our run-time to $O(n^2)$, since that j++ statement can run n x n = n^2 times.

Question 4

```
void function8(int n) {
        while (n > 0) {
                printf("%d\n", n);
                n = n/2;
        }
}
```

Solution
Each loop iteration, n is divided by 2 and we stop the step after n = 1 (because of integer division). After k loop iterations, the value of n is oldn/2^k
, where oldn represents the original value of n. Thus, we must have $n/2^k = 1$. Our goal is to find k, the number of iterations this code runs. Multiplying, we get $2^k = n$.
solving, by definition of log, we have $k = \log_2 n$. Thus, the runtime is O(lg n).

Question 5

```
int function9(int n) {
        int i,j;
```

```
        for (i=0; i<n; i++)
                for (j=0; j<n; j++)
                        if (j == 1)
                                break;
        return j;
}
```

Solution

The j loop never runs more than twice for each value of I, because it gets stopped at j = 1. Thus, the total number of times it runs is at most 2 x n. Thus, the function runs in O(n) time.

Question 6

Consider the following function with integer inputs n and m:

```
 void solveit(int* array, int n, int m)
 {
    int i, res = 0;
    for (i=0; i<n; i++)
    {
        int low = 0, high = m;
        while (low < high)
        {
            int mid = (low+high)/2;
            if (f(mid) < array[i])
                low = mid+1;
            else
               high = mid-1;
        }
        printf("%d\n", low);
    }
}
```

You may assume that the function f that is called from solveit defines a function that runs in O(1) time. With proof, determine the run-time of this function in terms of n and m.

Solution:

The outer loop runs n times. The inner loop runs a binary search between 0 and m, subdividing that interval by 2 with each loop iteration. The most number of times this loop could run is roughly lg m. (A more formal proof simply let's k equal the number of times the loop runs and solves for k in the equation $m/ 2^k = 1$. Multiply both sides to get $m = 2^k$. By definition of log, $k = log\ m$.) It follows that the run time is O(nlgm).

Question 7:

Below is a program which includes a single function call to the function mysqrt. The function mysqrt includes a while loop. Give an estimate and analysis on how many times that while loop will run during the single function call from main.

```
int main() {
    printf("%.6f\n", mysqrt(1000));
    return 0;
}
double mysqrt(double n) {
    double low = 1, high = n;
    if (n < 1) {
          low = n;
          high = 1;
    }
    while (high - low > .000001) {
        double mid = (low+high)/2;
        if (mid*mid < n)
                  low = mid;
      else
            high = mid;
   }
}
```

Solution:
The difference between high and low at the beginning of the function call is 1000 – 1 = 999. For the purposes of our rough analysis, we can simply call this 1000. The while loop repeatedly divides this difference by 2 for each iteration. Thus, after k iterations the difference between high and low is 1000/2 . Thus, we want the minimum value of k such that:

$$\frac{1000}{2^k} < 10^{-6}$$
$$\frac{10^3}{10^{-6}} < 2^k$$

$$2^k > 10^9$$
$$k > log_2 10^9$$

This value is irrational, but we can give an approximation for 10^9 in terms of a power of 2. Roughly speaking 2^{10} (1024) is fairly close to $1000 = 10^3$. So, to get a reasonable rough estimate, we can substitute 2^{30} for 10^9 , which will lead to the value of $k \sim 30$

Question 8:

What would be the worst case runtime for the push and pop function for a stack?

Answer: O(1) as the push and pop operation does not need to iterate through the items. It directly access the top item.

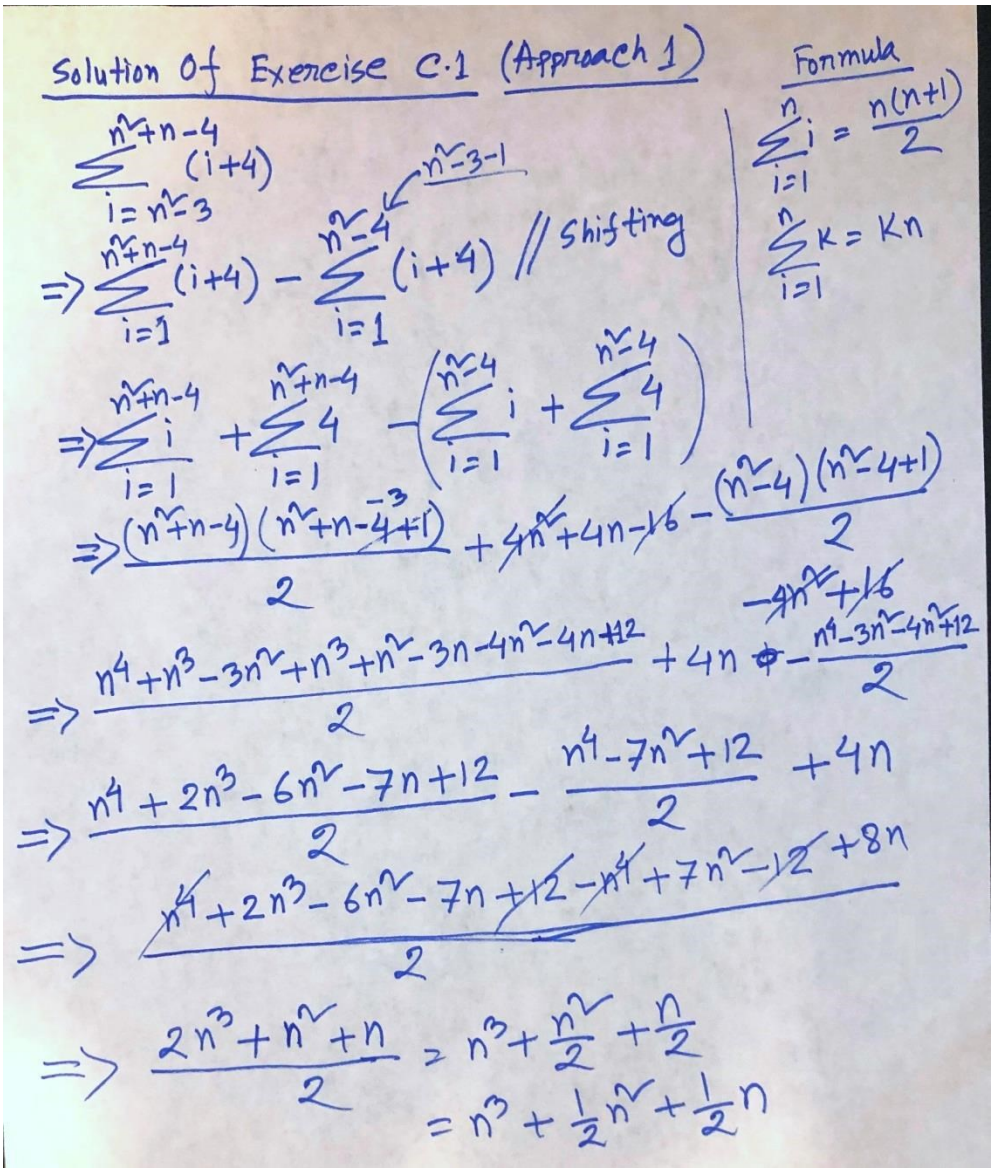
Part C Summation:

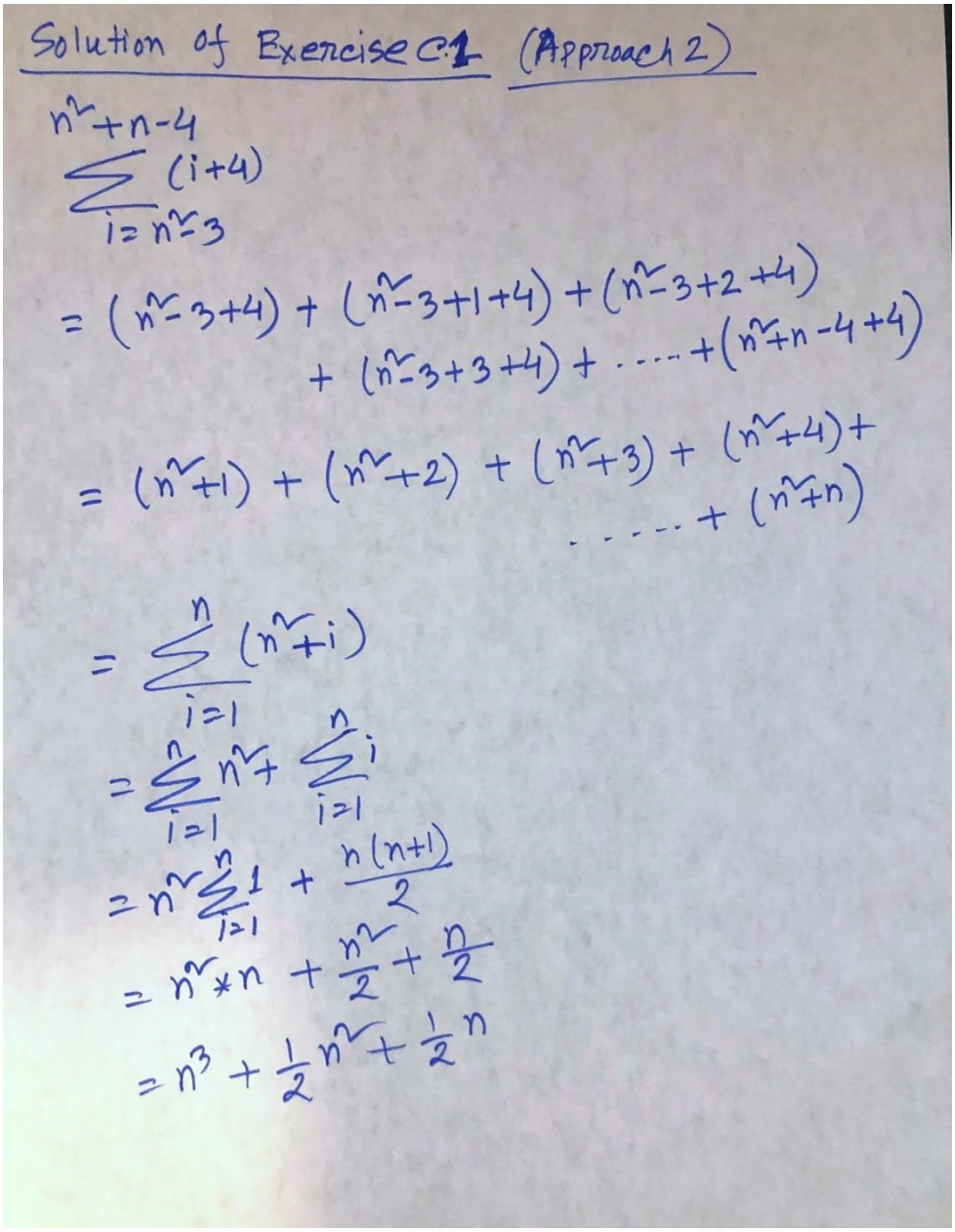
Before starting the summation exercises, go through the examples in the slides and try to solve them without looking at the solution if you think any example challenging to you.

Question 1

Determine the following summation in terms of n (assume n is a positive integer 2 or greater), expressing your answer in the form $an^3 + bn^2 + cn$, where a, b and c are rational numbers. (Hint: Try rewriting the summation into an equivalent form that generates less algebra when solving.)

$$\sum_{i=n^2-3}^{n^2+n-4} (i+4)$$





Question 2

1. There is a formula: $\sum_{i=0}^{n-1} 2^i = 2^n - 1$

(a) Using this result, determine a closed-form solution in terms of *n*, for the summation below.

(b) Determine the numeric value of the summation for n = 9.

$$\sum_{i=0}^{n} (\sum_{j=0}^{i-1} 2^j)$$

$$\sum_{i=0}^{n}(\sum_{j=0}^{i-1}2^j)$$

$$\sum_{i=0}^{n}(\sum_{j=0}^{i-1}2^j) = \sum_{i=0}^{n}(2^i - 1)$$

$$= \sum_{i=0}^{n}2^i - \sum_{i=0}^{n}1$$

$$= 2^{n+1} - 1 - (n+1)$$

$$= 2^{n+1} - n - 2$$

ANS(b):

$2^{9+1} - 9 - 2 = 1024 - 11 = \mathbf{1013}$

Question 3

1. Let *a, b*, *c,* and *d*, be positive integer constants with $a < b$. Prove that

$$\sum_{i=a}^{b}(ci + d) = \frac{(c(a+b) + 2d)(b - a + 1)}{2}$$

$$\sum_{i=a}^{b}(ci+d)=\sum_{i=1}^{b}ci-\sum_{i=1}^{a-1}ci+\sum_{i=a}^{b}d$$
$$=\frac{cb(b+1)}{2}-\frac{c(a-1)a}{2}+(b-a+1)d$$
$$=\frac{c(b^2+b-(a^2-a))}{2}+\frac{2d(b-a+1)}{2}$$
$$=\frac{c(b^2+b-a^2+a)}{2}+\frac{2d(b-a+1)}{2}$$
$$=\frac{c(b^2-a^2+b+a)}{2}+\frac{2d(b-a+1)}{2}$$
$$=\frac{c((b-a)(b+a)+(b+a))}{2}+\frac{2d(b-a+1)}{2}$$
$$=\frac{c(b+a)(b-a+1)}{2}+\frac{2d(b-a+1)}{2}$$
$$=\frac{(c(a+b)+2d)(b-a+1)}{2}$$

The key to this proof is setting up the usual summation formulas and then recognizing that b^2 - a^2 factors. Once this term is factorized, it can be recognized that (a+b) is a factor in the first term that can be pulled out. This, in turn, reveals (b - a + 1) to be a factor between the two larger terms, something that could have been anticipated since (b - a + 1) is a factor in the result on the RHS and also an immediate factor in the last term after the initial algebra.