COMP 3804/Math 3804: Assignment 1

Due Date: Sunday, October 2^{nd} at 11:59PM

Ryan Lo (101117765)

School of Computer Science Carleton University

Your assignment should be submitted online on Brightspace as a single .pdf file. The filename should contain your name and student number. No late assignments will be accepted. You can type your assignment or you can upload a scanned copy of it. Please, use a good image capturing device. Make sure that your upload is clearly readable. If it is difficult to read, it will not be graded.

Question 1 [15 marks]

Consider the following (not so great) algorithm defined on a set S of n numbers; assume that $n = 3k + 1$, for some non-negative integer k.

```
For i = 1 to ⌊(n/3)⌋ do
    STEP 1: in linear time (in the size of S), find and then delete both min- and max-element
        from S
    STEP 2: Again, in linear time, find and then delete from S (as updated by STEP 1)
        (just) the max-element.
```

I. First, what does this algorithm do more precisely? (It might help you, to first think about what the algorithm does if you were to omit Step 2. Do **not** write this as part of your solution.)

   The algorithm goes through a set of n numbers and finds both the min and the max element then removes it from the set. After it finishes that, it goes through the set again and finds only the max element then removes it from the set.

II. Then, write this algorithm in more detailed pseudo-code (filling in the min and max finding, in particular). Also make sure that you take care of the boundary/terminating conditions.

```
If n < 3 then return

For i = 1 to n/3,

        Min = S[0], max = S[0], min_index = 0, max_index = 0

        For j = 1 to length (S),

                If S[j] < min then min = S[j], min_index = j

                If S[j] > max then max = S[j], max_index = j

        Delete S[min_index]

        Delete S[max_index]

        S.length -= 2
```

```
max = S[0], max_index = 0
For j = 1 to length (S),
    If S[j] > max then max = S[j], max_index = j
Delete S[max_index]
S.length --
```

III. What is the time complexity of this (bad) algorithm?

It goes through the whole set of n elements to find the min and max for step 1, it goes through the whole set of n-2 elements again to find the max. It creates new set with n-3 elements.

$$O(3n) = O(n)$$

IV. Then, prove that this algorithm correctly finds that element of a set of *n* elements as determined by you in Part I.

Input: S[n] a set of n numbers

Output: S[n-3] a set of n numbers with the min and two max numbers removed

Loop invariant: At the start of iteration j of the loop, the variable answer should contain the subarray S[0:n-3] with the min and two max numbers removed.

Initialization: At the start of the first iteration of the loop, the variable answer should contain the first number as both the min and the max number.

Maintenance: Assume that the loop invariant holds at the start of iteration j. At the start of iteration j + 1, the answer will change depending on the number of j + 1 after comparing it to the current min and max.

Termination: When the loop terminates, the variable answer contains the subarray with the min and two max numbers removed.

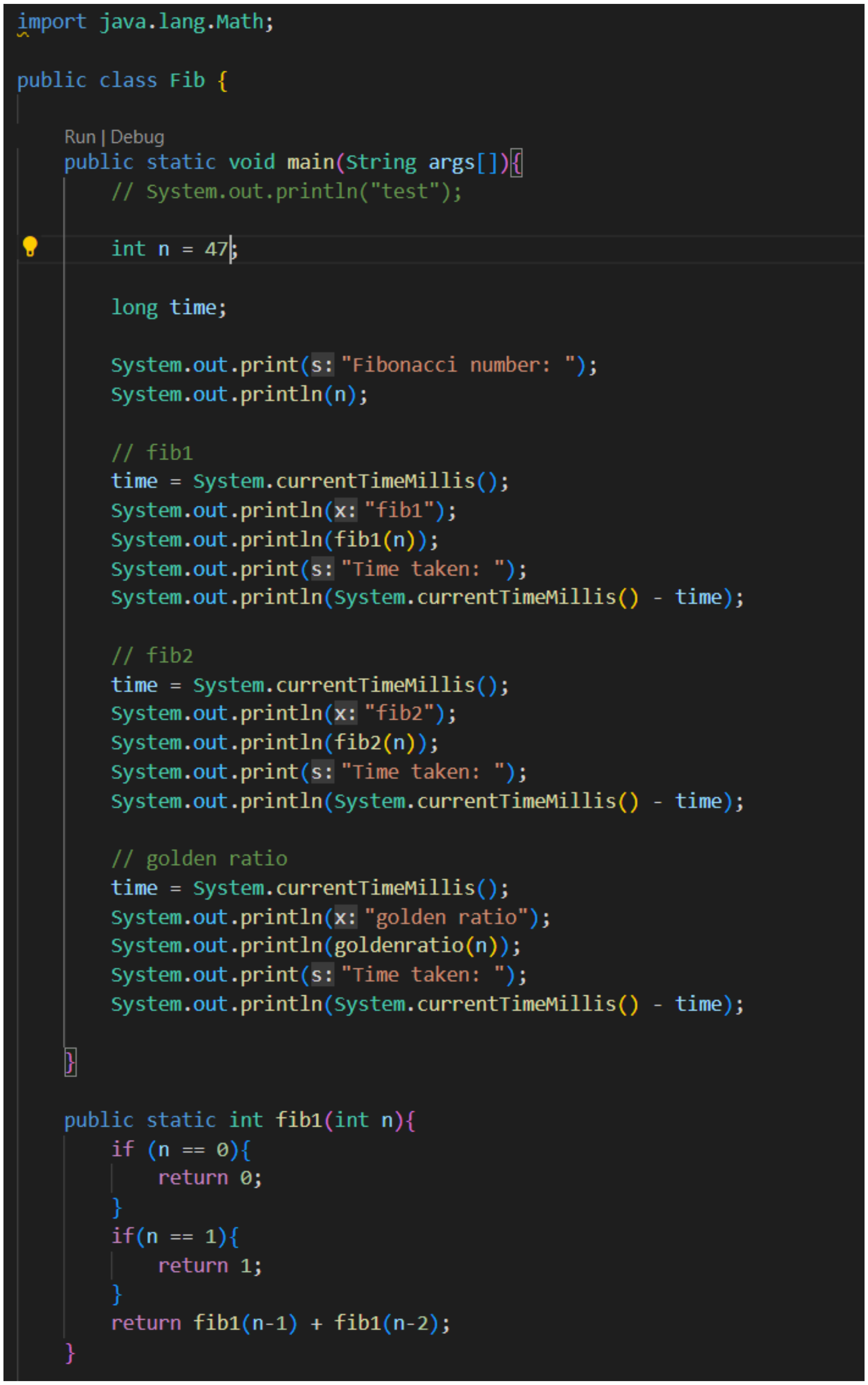
Question 2 [11 marks]

Implement the two algorithms presented in class for computing Fibonacci numbers, fib1 and fib2 and the direct way using the Golden Ratio.

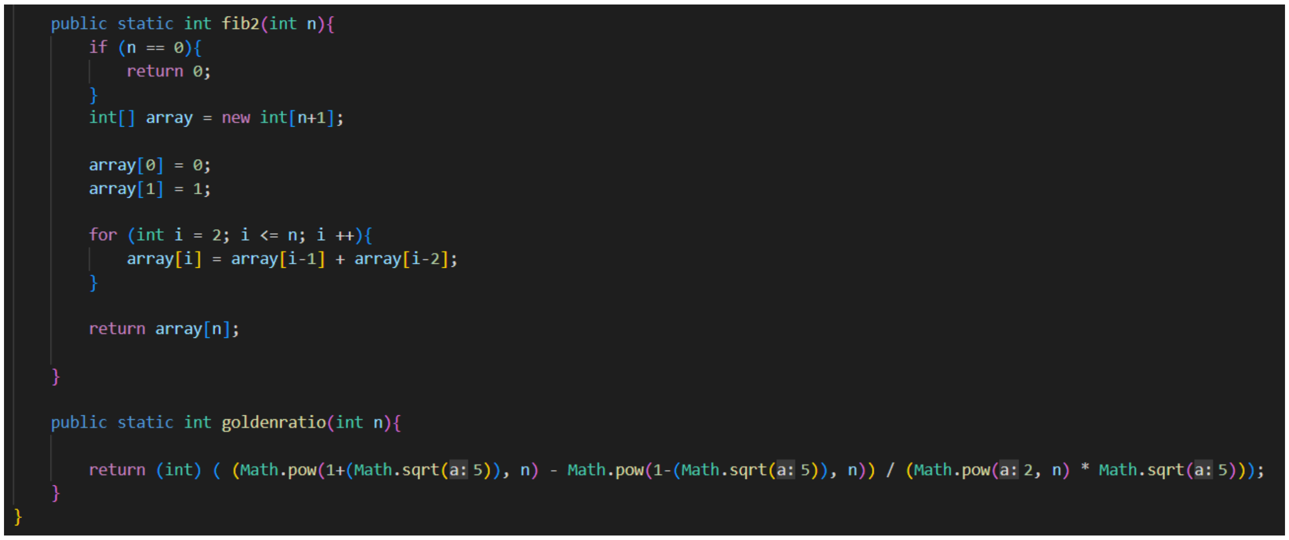
Each time you compute a Fibonacci number, measure how many milliseconds it takes, and print this timing information. There is a Java function, System.currentTimeMillis(), which returns a long result; call it once before you do the computation, and a second time after you do the computation; subtract to get the elapsed time. Be careful to measure only the computation time; any changes to the GUI should be made either before or after you start measuring. Do not do anything else on your computer. From time to time, your computer will do garbage collection, this may explain strange timing results.

What is the maximum value of *n* for which you can compute Fibonacci? How long does it take for each *n* that you can compute? Hand in a listing of your code and the timing results.

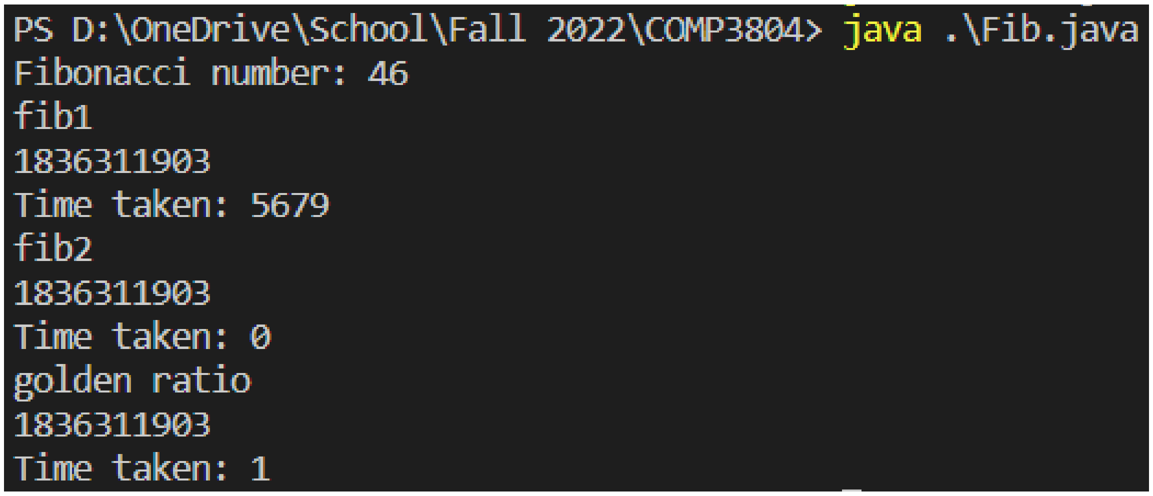
Using int to store the variable, the maximum value of n that I can compute Fibonacci was 46. Any Fibonacci number above 46 made the result return negative as a result of integer overflow. It took about 5679 ms for fib1 to complete. The other two ran almost instantly.



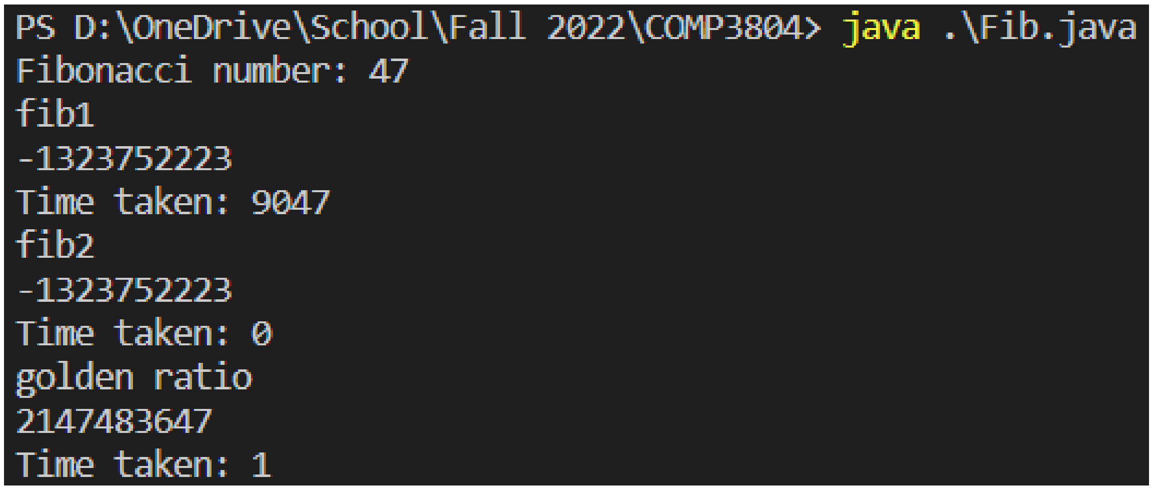
```java
import java.lang.Math;

public class Fib {

    Run | Debug
    public static void main(String args[]){
        // System.out.println("test");

        int n = 47;

        long time;

        System.out.print(s: "Fibonacci number: ");
        System.out.println(n);

        // fib1
        time = System.currentTimeMillis();
        System.out.println(x: "fib1");
        System.out.println(fib1(n));
        System.out.print(s: "Time taken: ");
        System.out.println(System.currentTimeMillis() - time);

        // fib2
        time = System.currentTimeMillis();
        System.out.println(x: "fib2");
        System.out.println(fib2(n));
        System.out.print(s: "Time taken: ");
        System.out.println(System.currentTimeMillis() - time);

        // golden ratio
        time = System.currentTimeMillis();
        System.out.println(x: "golden ratio");
        System.out.println(goldenratio(n));
        System.out.print(s: "Time taken: ");
        System.out.println(System.currentTimeMillis() - time);

    }

    public static int fib1(int n){
        if (n == 0){
            return 0;
        }
        if(n == 1){
            return 1;
        }
        return fib1(n-1) + fib1(n-2);
    }
```



```java
    public static int fib2(int n){
        if (n == 0){
            return 0;
        }
        int[] array = new int[n+1];

        array[0] = 0;
        array[1] = 1;

        for (int i = 2; i <= n; i ++){
            array[i] = array[i-1] + array[i-2];
        }

        return array[n];

    }

    public static int goldenratio(int n){

        return (int) ( (Math.pow(1+(Math.sqrt(a: 5)), n) - Math.pow(1-(Math.sqrt(a: 5)), n)) / (Math.pow(a: 2, n) * Math.sqrt(a: 5)));
    }
}
```



```
PS D:\OneDrive\School\Fall 2022\COMP3804> java .\Fib.java
Fibonacci number: 46
fib1
1836311903
Time taken: 5679
fib2
1836311903
Time taken: 0
golden ratio
1836311903
Time taken: 1
```



```
PS D:\OneDrive\School\Fall 2022\COMP3804> java .\Fib.java
Fibonacci number: 47
fib1
-1323752223
Time taken: 9047
fib2
-1323752223
Time taken: 0
golden ratio
2147483647
Time taken: 1
```

Question 3 [34 marks]

(See also the Tutorial on recurrences)

For the following we will assume that T(1) =1. (If required, you can also assume that T(*k*) = O(1), for any constant *k*.) For 1-3, "solve" means provide a big-oh bound.

1. Solve, by unfolding also known as the iterative method (or repeatedly applying the recurrence relation):

   a. $T(n) = T(n-2) + n$

   $$T(n) = (T(n-2-2) + n-2) + n$$

   $$T(n) = (T(n-4) + n-2) + n$$

   $$T(n) = T(n-4) + 2n - 2$$

   $$T(n) = T(n-2k) + kn - k$$

   $$T(n) = 1 + kn - k = O(n)$$

   b. $T(n) = 2T(n/4) + n$.

   $$T(n) = 2T\left(\frac{n}{4}\right) + n$$

   $$T(n/4) = 2T\left(\frac{n}{4^2}\right) + \frac{n}{4}$$

   $$T(n) = 2(2T\left(\frac{n}{4^2}\right) + \frac{n}{4}) + n$$

   $$T(n) = 2^2T\left(\frac{n}{4^2}\right) + \frac{n}{2} + n$$

   $$T(n) = 2^2(2T\left(\frac{n}{4^3}\right) + \frac{n}{4^2}) + \frac{n}{2} + n$$

   $$T(n) = 2^3T\left(\frac{n}{4^3}\right) + 2^2\frac{n}{4^2} + \frac{n}{2} + n$$

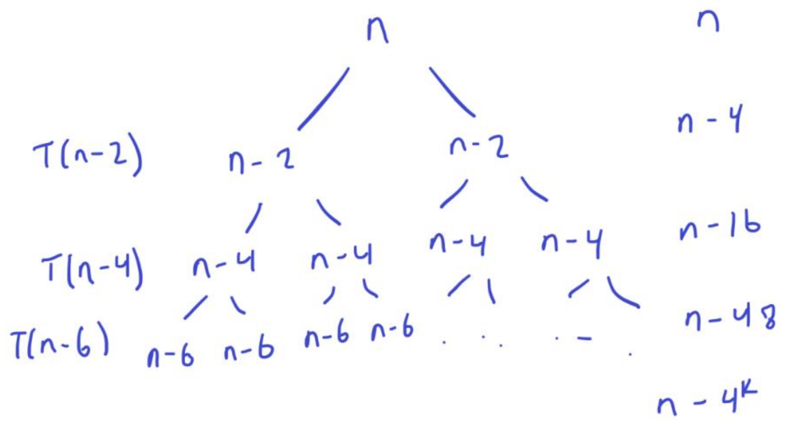
   $$T(n) = 2^{\mathrm{k}}T\left(\frac{n}{4^{\mathrm{k}}}\right) + \mathrm{k}\frac{n}{4} + n$$

   $$\frac{n}{4^k} = 1\,, n = 2k\,, \log_4 n = \log_4 2^k\,, k = \log_4 n$$

   $$= n * 1 + \frac{1}{4} log_4 n = O(n\, log_4 n)$$

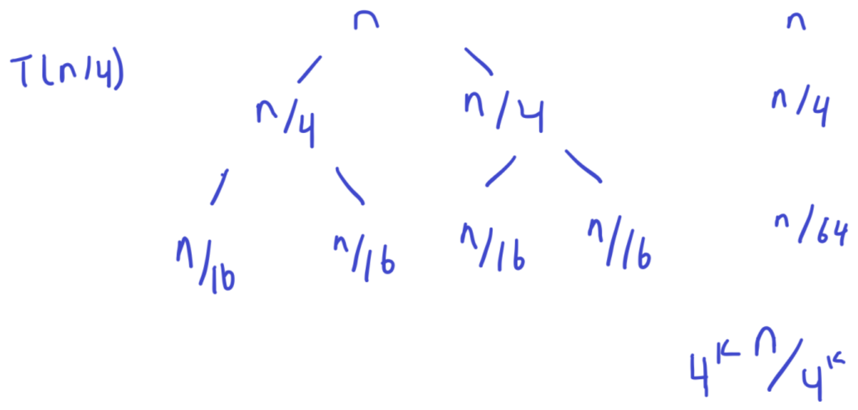
2. Now, solve by applying the recursion tree method:

   a. $T(n) = T(n - 2) + n$.



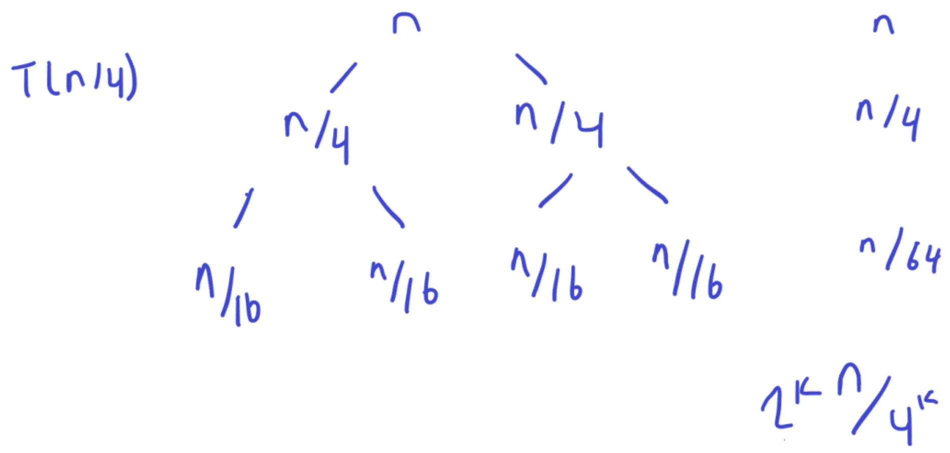
$T(n) = O(n)$

   b. $T(n) = 4T(n/4) + n$.



$T(n) = O(n\, log_4\, n)$

   c. $T(n) = 2T(n/4) + n$.



$$T(n) = O(n\,log_4\,n)$$

3. Next, solve by guessing the solution and then proving it using induction:
   a. $T(n) = 8T(n/8) + n$.

      Take a guess that $T(n) = O(n \log_8 n)$.

      Verify if $T(n) = O(n \log_8 n)$

      Need to check $T(n) \leq cn \log_8 n$ for some constant $c \geq 1$

      We will show that $T(n) = 8T\left(\frac{n}{8}\right) + n \leq cn \log_8 n$?

$T(n) = 8T\left(\frac{n}{8}\right) + n$

$T(n) = \mathrm{c}\; n/8 \log_8 n/8 + n$

$T(n) = \mathrm{c}n(logn - \log 8) + n$

$T(n) = \mathrm{c}nlogn - cn \log 8 + n$

$T(n) = \mathrm{c}nlogn - n(c \log 8 + 1)$

$T(n) = nlogn - n(\log 8 + 1),\; c = 1$

$T(n) = nlogn - n(\log 8 + 1) \leq \boldsymbol{nlogn}$

$T\ (n)\ =\ 8T\ (n/8)\ +\ n$

$T\ (n)\ =\ 8(8T\ (n/8^2)\ +\ n/8) +\ n$

$T\ (n) =\ 8^2\ T\ (n/8^2)\ +\ n +\ n$

$T\ (n) =\ 8^2\ T\ (n/8^2)\ +\ 2n$

$T\ (n) =\ 8^{\mathrm{k}}\ T\ (n/8^{\mathrm{k}})\ +\ kn$

$T\ (n) =\ n * 1\ +\ n \log_8 n = O(n \log_8 n)$

b. $T(n) = T(n/2) + log\ n$.

Take a guess that $T(n) = O(log\, n)$.

Verify if $T(n) = O(log\, n)$

Need to check $T(n) \leq c\, log\, n$ for some constant $c \geq 1$

We will show that $T(n) = T\ (n/2)\ +\ log\ n \leq c\, log\, n$?

$$T(n) = c\ (log\, n\, /2)\ +\ log\ n$$
$$T(n) = c\ (log\, n - \log 2)\ +\ log\ n$$
$$T(n) = c\ log\, n - \mathrm{c} \log 2\ +\ log\ n$$
$$T(n) = log\, n(c+1) - c\ log\ 2 \leq \mathrm{logn}$$

$$T\ (n) =\ T(n/2) + \log n$$
$$T\ (n) =\ T(n/4) + \log n/2 + \log n$$
$$T\ (n) =\ T(n/8) + \log n/2 + \log n/2 + \log n$$
$$T\ (n) =\ T\left(n/2^k\right) + \mathrm{k}\log n/2 + \log n = O(\log n)$$

c. $T(n) = T(n/2^n) + 1$.

Take a guess that $T(n) = O(logn)$.

Verify if $T(n) = O(logn)$

Need to check $T(n) \leq clogn$ for some constant $c \geq 1$

We will show that $T(n) = T\ (n/2n)\ +\ 1 \leq clogn$?

$$T\ (n) =\ T(n/2^n) + 1$$
$$T\ (n) =\ c\ log\ n/2^n\ + 1$$
$$T\ (n) =\ c\ (log\ n - log2^n)\ + 1$$
$$T\ (n) =\ c\ (log\ n - nlog2)\ + 1$$
$$T\ (n) =\ c\ log\ n - cnlog2\ + 1$$

$$T\ (n) =\ T(n/2^n) + 1$$
$$T\ (n) =\ T\left(\frac{n}{2^n * 2^{n/2^n}}\right) + 1 + 1$$
$$T\ (n) =\ T\left(\frac{n}{2^n * k2^{n/2^n}}\right) + k$$
$$T\ (n) =\ O(1)$$

4. Finally, apply the Master Theorem (in its general form), where possible, to solve the following recurrence relations and give a Θ-bound for each of them. If the master's theorem is not applicable, state why and then solve the recurrence using another method that we learned in class.

   a. $\boldsymbol{T(n) = T(n - 4) + n}$**.**

   The Master Theorem is not applicable because it is not in the form of T(n) = aT(n/b) + f(n). It would be easier to solve this recurrence using induction substitution.

$$T\ (n)\ =\ T\ (n\ -\ 4)\ +\ n$$

$$T\ (n)\ =\ \ (T\ ((n\ -\ 4) - 4)\ + (n - 4)) +\ n$$

$$T\ (n)\ =\ \ T\ (n\ -\ 8)\ + n - 4 +\ n$$

$$T\ (n) =\ \ T\ (n\ -\ 8) + 2n - 4$$

$$T\ (n) =\ \ T\ (n\ -\ 4k) + kn - k$$

$$T\ (n) = 1 + kn - k$$

$$T\ (n) =\ \ \theta(n)$$

b. $\boldsymbol{T(n) = T(9n/3) + n^2}$**.**

a = 1, b = 3/9 = 1/3, d = 2, f(n) = n^2

Calculate $n^{\log_b a} = n^{\log_{1/3} 1} = \theta(1)$

There is no $\varepsilon > 0$ that applies to this recurrence. The Master Theorem cannot be applied to this recurrence.

$$T(n) = T\ (9n/3)\ +\ n^2$$

$$T(n) = T\ (81n/9)\ +\ \left(\frac{9n}{3}\right)^2\ +\ n^2$$

$$T(n) = T\ (9^{\mathrm{k}}n/3^{\mathrm{k}})\ +\ \mathrm{k}\left(\frac{9n}{3}\right)^2\ +\ n^2$$

$$T(n) = 1\ +\ 81\mathrm{n}^2 +\ n^2 = O(n^2)$$

c. $\boldsymbol{T(n) = T(16n/4) + n^2}$**.**

a = 1, b = 4/16 = 1/4, d = 2

Calculate $n^{\log_b a} = n^{\log_{1/4} 1} = \theta(1)$

There is no $\varepsilon > 0$ that applies to this recurrence. The Master Theorem cannot be applied to this recurrence.

$$T(n) = T\ (16n/4)\ +\ n^2$$

$$T(n) = T\ (256n/16)\ +\ \left(\frac{16n}{4}\right)^2\ +\ n^2$$

$$T(n) = T\ (16^{\mathrm{k}}n/4^{\mathrm{k}})\ +\ \mathrm{k}\left(\frac{16n}{4}\right)^2\ +\ n^2$$

$$T(n) = 1\ +\ 256\mathrm{n}^2 +\ n^2 = O(n^2)$$

d. $T(n) = 7T(n/3) + n^2$.

a = 7, b = 3, d = 2

Calculate $n^{\log_b a} = n^{\log_3 7} = \theta(n^{1.7712})$

Since $f(n) = \Omega\left(n^{\log_3 7+\varepsilon}\right)$ with $\varepsilon = 2$, we are in case 3.

We need to show that $a * f\left(\frac{n}{b}\right) < cf(n)$ for some constant c < 1 and all sufficiently large n.

Then we get $T(n) = \theta(f(n) = \theta(n^2))$

$7 * \left(\frac{n}{3}\right)\log\left(\frac{n}{3}\right) \leq \left(\frac{7}{3}\right)n^2 = \left(\frac{7}{3}\right)f(n), c = \left(\frac{7}{3}\right)$

$\therefore T(n) = \theta(f(n) = \theta(n^2))$

e. $T(n) = 2^nT(n/2) + O(1)$

a = 2^n, b = 2, d = O(1)

Calculate $n^{\log_2 2^n} = n^n = \theta(n^n)$

f. $T(n) = cT(n/c) + n$ for some positive integer c.

a = c, b = c, d = n

if c = 1 then it doesn't terminate, otherwise

Calculate $n^{\log_b a} = n^{\log_c c} = \theta(n^1) = \theta(n)$

g. $\boldsymbol{T(n) = 2T(n/4) + n^{0.6}}$**.**

a = 2, b = 4, d = $n^{0.6}$

Calculate $n^{\log_b a} = n^{\log_4 2} = \theta(n^{0.5})$

Since $f(n) = \Omega\left(n^{\log_4 2+\varepsilon} =\right)$ with $\varepsilon = 0.3$, we are in case 3.

We need to show that $a * f\left(\frac{n}{b}\right) < cf(n)$ for some constant c < 1 and all sufficiently large n.

Then we get $T(n) = \theta(f(n) = \theta(n^{0.6}))$

$2 * \left(\frac{n}{4}\right) \log\left(\frac{n}{4}\right) \leq \left(\frac{2}{4}\right) n^2 = \left(\frac{2}{4}\right) f(n), c = (2/4) = (1/2)$

$\therefore T(n) = \theta(f(n) = \theta(n^{0.6}))$

5. a. What can you say about the time complexity of an algorithm whose running time is given by this recurrence $T(n) = 2T(n) + O(\log^2 n)$?

a = 2, b = 1, d = $\log^2 n$

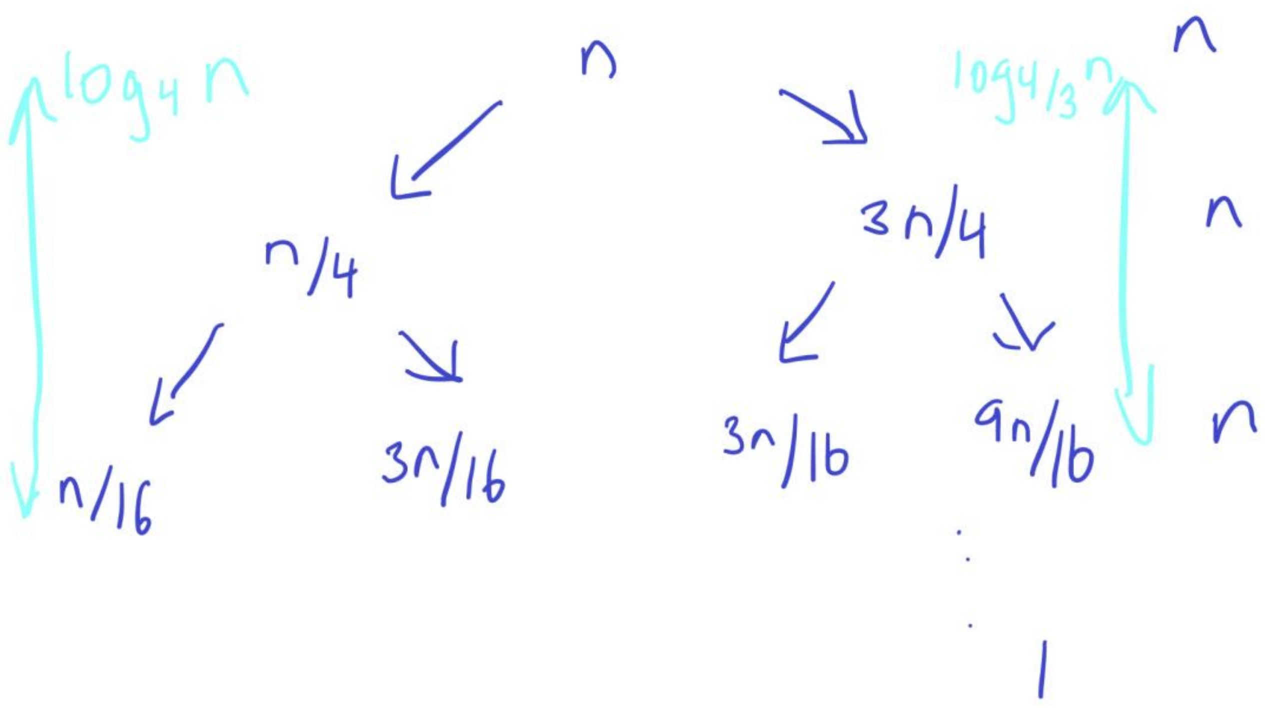
$\log^2 n < log\, 2\,?$

If $n\ <\ 3.6$ then $d < log\, 2$ and $T(n) = O(n^{\log_2 1})$

If $n >\ 3.6$ then $d > log\, 2$ and $T(n) = O(\log^2 n)$

b. Which method might be most appropriate to solve the following recurrence? Use it to give its solution. $T(n) = T(n/4) + T(3n/4) + n$

The method that would be most appropriate to solve this recurrence would be using the recursion tree method.



$$n \log_4 n \leq T(n) \leq n \log_{4/3} n$$

Question 4 [15 marks]

DT claims that he has a data structure that he can create in O(n) time and on which he can perform the operation DeleteMin using *o*(*logn*) comparisons. (note the little-oh)

(a) What could you say about the number of comparisons used by the most efficient sorting algorithm designed based on DT's claim? Argue.

o(logn) states that "For **every** choice of a constant k > 0, you can find a constant a such that the inequality 0 <= f(x) < k g(x) holds for all x > a."
O(logn) states that " For **at least one** choice of a constant k > 0, you can find a constant a such that the inequality 0 <= f(x) <= k g(x) holds for all x > a."

Based on BT's claim, he says that his DeleteMin operation will always be equal or faster than logn comparisons. This doesn't seem possible as you need to look through every element as least once and compare them.

(b) Based (a) argue that DT must be lying.

If the data structure is unsorted, there are always going to be n! outcomes. You will always need n comparisons. You will always need to look through every element as least once and compare them.

Question 5 [25 marks]

Informally, a convex polygon is a (non self-intersecting) polygon in which each interior angle is less than 180^o.

- Draw a convex polygon based on that definition.
- Draw a self-intersecting polygon.

The convex hull of a set of points *S* is the smallest convex polygon containing all points of *S* (no point of *S* is outside the convex hull; the vertices of the polygon are points from *S*). (Without handing this in, draw examples for yourself.)

- A property of the convex hull is that, for each edge *e* of the convex hull polygon, all points of *S* are on one side of the (infinite) line drawn through *e* (some points may be on the line). Draw this and hand in with your solution.

  Now, design a simple algorithm that uses this property. Prove its correctness and establish its time complexity.

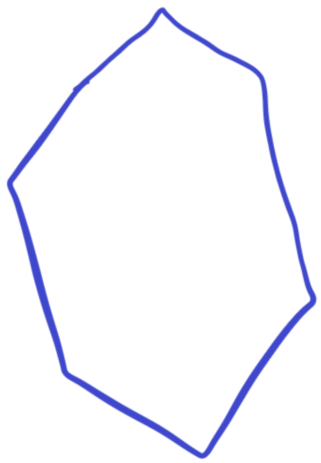
- Let us see if we can do better. You are given two linear-time algorithms which you can just use without having to give pseudocode nor prove their correctness. (Like use as a black box)

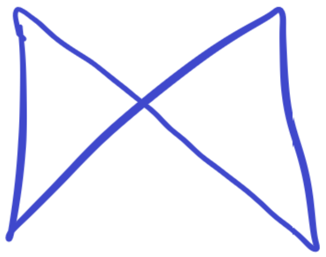
  The first one is a median finding algorithm. The median of a set of *n* numbers is the element for which $\lfloor(n/2)\rfloor$ are less than or equal to it and the remaining elements are larger than it.

  The second one computes the convex hull of two point sets for which we are given their convex hulls. The points sets here are assumed to be separated by a vertical line. See Figure 1.

Now use this to design a D&C algorithm to find convex hull of a set on *n* points. Prove its correctness and establish its time complexity.

Pick a starting point, Loop through everything till you find all the point on the hull, returns true if all on one side false if not





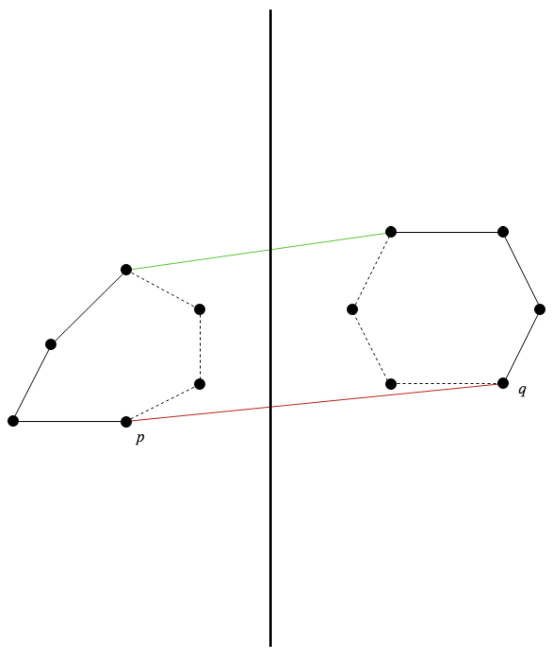


Figure 1: For Question 5: Merging two Convex Hulls separted by a vertical line