EECS 281, Week 2: Monday

Hi again! Slides, code examples at `maximal.io/eecs183` *! Notes at* `datastructures.maximal.io`.

Binomial coefficient

Recall that the binomial coefficient can be written as $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

Implement `binomial`*, a function that computes the Binomial coefficient of* `n` *and* `k`.

```
uint64_t binomial(int n, int k) {




}
```

Asymptotics

Name	Big O	Big Omega	Big Theta
Notation	$f(n) \in O\left((n)\right)$	$f(n) \in \Omega\left((n)\right)$	$f(n) \in \Theta\left((n)\right)$
Informal meaning	Order of growth $f(n)$	Order of growth $f(n)$	Order of growth $f(n)$
Example family	$O(N^2)$	$\Omega(N^2)$	$\Theta(N^2)$
Family members	$\frac{N^2}{2}$ $2N^2 + 1$ $\log(N)$	$\frac{N^2}{2}$ $2N^2 + 1$ e^N	$\frac{N^2}{2}$ $2N^2 + 1$ $N^2 + 183N + 5$

Order from most to least efficient:

$O(n)$ $O(n!)$ $O(n^n)$ $O(n^2)$ $O(2^n)$ $O(\log n)$ $O(1)$
$O\left(\sqrt{n}\right)$ $O(n^3)$ $O(n^2 \log n)$ $O(3^n)$ $O(n \log n)$

_____ ⊂ _____ ⊂ _____ ⊂ _____ ⊂ _____ ⊂ _____ ⊂ _____ ⊂

_____ ⊂ _____ ⊂ _____ ⊂ _____ ⊂ _____.

EECS 281, Week 2: Monday

Analysis of Algorithms (1)

Let $R(N)$ be the runtime of this code as a function of N, where N is the size of the array.

```
bool hasDuplicates(const int numbers[], int size) {
    for (int i = 0; i < size; i += 1) {
        for (int j = i + 1; j < size; j += 1) {
            if (numbers[i] == numbers[j]) {
                return true;
            }
        }
    }
    return false;
}
```

What is the order of growth of $R(N)$? _____

Find a simple $f(N)$ such that the runtime $R(N) \in \Theta(f(N))$ in the worst case. _____

Best case vs. Worst case

Best case: _____

Average case: _____

Worst case: _____

Always: _____

Analysis of Algorithms (2)

Assume that `mysteryFunc` executes in constant time and returns a value of type `int`.

```
int j = 0;
for (int i = N; i > 0; i -= 1) {
    for (; j < M; j += 1) {
        if (mysteryFunc(i, j) > 0) {
            break;
        }
    }
}
```

Give the worst-case and best-case runtime in terms of N and M.

Worst-case: _____ Best-case: _____

EECS 281, Week 2: Monday

Analysis of Algorithms (3)

```
void printHello(int n) {
    for (int i = 1; i <= n; i *= 2) {
        for (int j = 0; j < i; j += 1) {
            cout << "hello" << endl;
        }
    }
}
```

Find a simple $f(N)$ such that the runtime $R(N) \in \Theta(f(N))$. _____

Formulas

Sum of first N numbers: $1 + 2 + 3 + 4 + \ldots + N =$ _____

Sum of first N powers of 2: $1 + 2 + 4 + 8 + \ldots + N =$ _____

Asymptotics

True or false? If $f(n) \in O(n^2)$ and $g(n) \in O(n)$ are positive-valued functions, then $\frac{f(n)}{g(n)} \in O(n)$.

True or false? If $f(n) \in \Theta(n^2)$ and $g(n) \in \Theta(n)$ are positive-valued functions, then $\frac{f(n)}{g(n)} \in \Theta(n)$.

Analysis of Algorithms (4) / Recursion

```
int recursive(int n) {
    if (n <= 1) {
        return 1;
    }
    return recursive(n - 1) + recursive(n - 1);
}
```

Find a simple $f(N)$ such that the runtime $R(N) \in \Theta(f(N))$. _____

EECS 281, Week 2: Monday

Recurrence Relations

There are three main ways to solve recurrence relations:

1. _____
2. _____
3. _____

Substitution Method

Steps:

1. Guess the form of the solution
2. Verify by induction
3. Solve for constants

$T(n) = 2T(n-1) + 1$

Recursion Tree Method

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

Master Theorem

Solve recurrences of the form _____ where $a \geq 1$, $b > 1$ and f is asymptotically positive. If $f(n) \in O(n^c)$, then

$$T(n) \in \begin{cases} & \text{if} \\ & \text{if} \\ & \text{if} \end{cases}$$

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

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

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

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

Improving Sorting

Assume that `mysteryFunc` executes in constant time and returns a value of type `int`.

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j − 1] > a[j])
        swap a[j] and a[j − 1]
        −−j
```

What is the growth complexity of this algorithm? _____

How can we improve this algorithm?

Merge

Implement `merge`, a function that combines the elements of two sorted vectors into a single vector and returns the result. For example, if `a` contains `{1, 2, 4}` and `b` contains `{2, 3, 5}`, `merge` should return a vector containing `{1, 2, 2, 3, 4, 5}`.

```
vector<int> merge(const vector<int> &a, const vector<int> &b) {









}
```

When finished, take a picture of your code and email it to Maxim.