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 $\binom{n}{k} = \frac{n!}{k!(n-k)!} = \binom{n-1}{k-1} + \binom{n-1}{k}$.

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

```
uint64_t binomial(int n, int k) {
    if (k > n) {
        return 0;
    } else if (k == 0 || n == k) {
        return 1;
    } else {
        return binomial(n - 1, k - 1) + binomial(n - 1, k);
    }
}
```

Asymptotics

Name	Big O	Big Omega	Big Theta
Notation	$f(n) \in O((n))$	$f(n) \in \Omega((n))$	$f(n) \in \Theta((n))$
Informal meaning	Order of growth is less than or equal to $f(n)$	Order of growth is equal to $f(n)$	Order of growth is greater than or equal to $f(n)$
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:

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

$O(1) \subset O(\log n) \subset O(\sqrt{n}) \subset O(n) \subset O(n \log n) \subset O(n^2) \subset O(n^2 \log n) \subset$
$O(n^3) \subset O(2^n) \subset O(3^n) \subset O(n!) \subset O(n^n)$.

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 $R(N)$? ________________depends on the input ______________

Find a simple $f(N)$ such that the runtime $R(N) \in \Theta(f(N))$ in the worst case. ________N^2 ________

Best case vs. Worst case

Best case: describes the performance of an algorithm under optimal conditions.aaa defined by the minimum number of steps taken on any instance of size *n*.

Average case: defined by the average number of steps taken on any instance of size *n*.

Worst case: defined by the maximum number of steps taken on any instance of size *n*.

Always: describes the performance of an algorithm that does not depend on the input.

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: _________$\Theta(N + M)$ _________ Best-case: ______________$\Theta(N)$______________

`j` is only initialized outside of the outer for loop.

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 $R(N) \in \Theta(f(N))$. ________________ N ________________

Formulas

Sum of first N numbers: $1 + 2 + 3 + 4 + \ldots + N = N(N + 1) / 2 \in \Theta(N^2)$ ______________

Sum of the powers of 2 up to N: $1 + 2 + 4 + 8 + \ldots + N = 2N - 1 \in \Theta(N)$ ________________

Asymptotics

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

False. Let $f(n) = n^2$ and $g(n) = \frac{1}{n}$, then $\frac{f(n)}{g(n)} = n^3$.

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

True.

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 $R(N) \in \Theta(f(N))$. ________________ 2^N ________________

An easy way is to construct a recurrence relation, $T(n) = 2T(n-1) + 1$, which we can solve with the substitution method.

EECS 281, Week 2: Monday

Recurrence Relations

There are three main ways to solve recurrence relations:

1. Substitution Method
2. Recursion Tree Method
3. Master Theorem

Substitution Method

Steps:

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

$T(1) = 1$

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

We can guess that the solution will be $O(2^n)$, since on each step n, we do $n-1$ units of work twice. We will try to prove that $T(n) \leq k2^n - b$ (we include b in anticipation of having to deal with $+1$ in the recurrence relation).

Now, we prove $T(n) \leq k2^n - b$ by induction.

Base case: $n = 1$. $T(1) = 1 \leq k2^1 - b = 2k - b$. This is true for any $k \geq \dfrac{b+1}{2}$.

Inductive step: we assume the property is true for $n-1$; we need to show that it is true for n.

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &\leq 2(k2^{n-1} - b) + 1 \\ &= k2^n - 2b + 1 \\ &\leq k2^n - b \end{aligned}$$

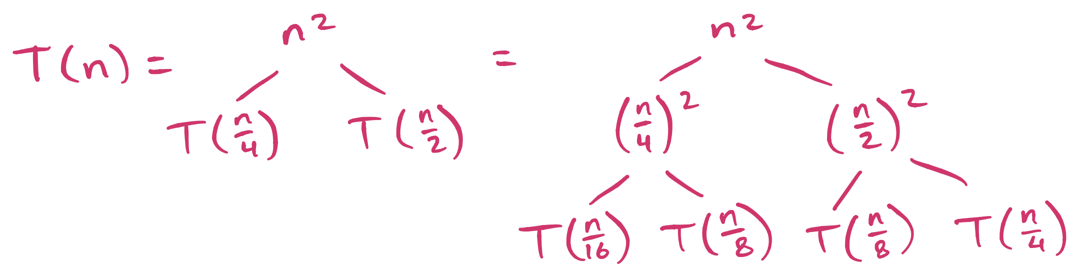
This is true for any $b \geq 1$.

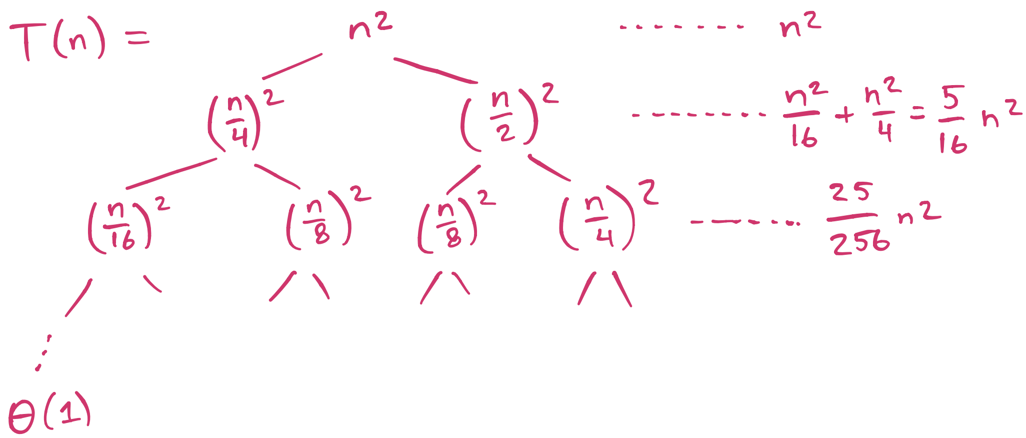
We can choose any constants b and k that satisfy $b \geq 1$ and $k \geq \dfrac{b+1}{2}$, such as $b = 1$ and $k = 1$.

Therefore, we have proved that the property is true and that $T(n) \in O(2^n)$.

Recursion Tree Method

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





$$\text{Total} = \underbrace{\left(1 + \frac{5}{16} + \frac{25}{256} + \dots + \frac{5^k}{16^k}\right)}_{< 2 \text{ since } 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots = 2} n^2 < 2n^2 \rightarrow O(n^2)$$

Master Theorem

Solve recurrences of the form $T(n) = aT\left(\frac{n}{b}\right) + f(n)$ where $a \geq 1, b > 1$ and f is asymptotically positive. If $f(n) \in O(n^c)$, then

$$T(n) \in \begin{cases} \Theta\left(n^{\log_b a}\right), & \text{if } a > b^c \\ \Theta\left(n^c \log n\right), & \text{if } a = b^c \\ \Theta\left(n^c\right), & \text{if } a < b^c \end{cases}$$

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

$a = 4$ $\quad$ $a \,?\, b^c$
$b = 2$ $\quad$ $4 > 2^1$ $\rightarrow$ case 1 $\rightarrow \Theta(n^{\log_2 4}) = \Theta(n^2)$
$c = 1$

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

$a = 4$ $\quad$ $a \,?\, b^c$
$b = 2$ $\quad$ $4 = 2^2$ $\rightarrow$ case 2 $\rightarrow \Theta(n^2 \log n)$
$c = 2$

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

$a = 4$ $\quad$ $a \,?\, b^c$
$b = 2$ $\quad$ $4 < 2^3$ $\rightarrow$ case 3 $\rightarrow \Theta(n^3)$
$c = 3$

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

Master Theorem does not apply

EECS 281, Week 2: Monday

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? ________ $O(N^2)$ ________

How can we improve this algorithm?

0 | 1 | 2 | 3 | 4 | 10 | 11

EECS 281, Week 2: Monday

Merge

i: 7 | 4 | 10 | 11

j: 0 | 2 | 3

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) {
    vector<int> result;
    result.reserve(a.size() + b.size());
    size_t i = 0;
    size_t j = 0;
    while (i < a.size() && j < b.size()) {
        if (a[i] < b[j]) {
            result.push_back(a[i]);
            ++i;
        } else {
            result.push_back(b[j]);
            ++j;
        }
    }
    while (i < a.size()) {
        result.push_back(a[i]);
        ++i;
    }
    while (j < b.size()) {
        result.push_back(b[j]);
        ++j;
    }

    return result;
}
```

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