CS3334 - Data Structures
Lab 2

Outline

- Program Complexity Exercises
- Assignment 819 Josephus Problem
- Assignment 744 Stack Shuffling

Exercise 1

- You are given 30 integers as the input. Write a procedure to output the largest value and the smallest integer value; and analyze the time complexity.

  n
- Suppose a[0] to a[29] are storing these integers.

  a[n]

Exercise 1

- You are given 30 integers as the input. Write a procedure to output the largest value and the smallest integer value.
- Suppose a[0] to a[29] are storing these integers.

Set two references min and max, make them equal to a[0] at the beginning.
For all the following integers a[1] to a[29],

- compare it with the current min and max,
- update min and max accordingly.

Note: Comparation number

for i in range(1, n)

```
if (a[i] > max)
    max = a[i]
if (a[i] < min)
    min = a[i]
```

$\Rightarrow n-1$

$\Rightarrow n-1$

$\Rightarrow 2\cdot(n-1)$

Exercise 1

- You are given 30 integers as the input. Write a procedure to output the largest value and the smallest integer value.
- Suppose a[0] to a[29] are storing these integers.

Set two references min and max, make them equal to a[0] at the beginning.
For all the following integers a[1] to a[29],

- compare it with the current min and max,
- update min and max accordingly.

```
if (a[i] > max)
      . . .
if (a[i] < min)
      . . .
```

2*(N-1)

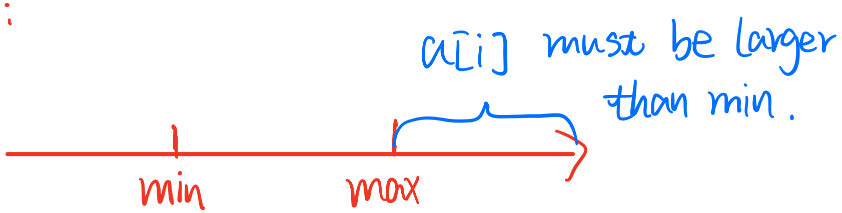
Program Complexity Exercise 1

- You are given 30 integers as the input. Write a procedure to output the largest value and the smallest integer value.
- Suppose a[0] to a[29] are storing these integers.

Set two references min and max, make them equal to a[0] at the beginning.
For all the following integers a[1] to a[29],

- compare it with the current min and max,
- update min and max accordingly.

Insight :




```
if (a[i] > max)
      . . .
if (a[i] < min)
      . . .
```

Program Complexity Exercise 1

- You are given 30 integers as the input. Write a procedure to output the largest value and the smallest integer value.
- Suppose a[0] to a[29] are storing these integers.

Set two references min and max, make them equal to a[0] at the beginning.
For all the following integers a[1] to a[29],

- compare it with the current min and max,
- update min and max accordingly.

$\lesssim 2 \cdot (n-1)$

```
if (a[i] > max)
      . . .
if (a[i] < min)
      . . .
```

```
if (a[i] > max)
      . . .
else if (a[i] < min)
      . . .
```

Program Complexity Exercise 1

- You are given 30 integers as the input. Write a procedure to output the largest value and the smallest integer value.
- Suppose a[0] to a[29] are storing these integers.

Set two references min and max, make them equal to a[0] at the beginning.
For all the following integers a[1] to a[29],

- compare it with the current min and max,
- update min and max accordingly.

<= 2*(N-1)

```
if (a[i] > max)
      . . .
if (a[i] < min)
      . . .
```

→

```
if (a[i] > max)
      . . .
else if (a[i] < min)
      . . .
```

Program Complexity Exercise 1

- You are given 30 integers as the input. Write a procedure to output the largest value and the smallest integer value.
- Suppose a[0] to a[29] are storing these integers.

Do pairs of comparisons to decide which half of numbers will produce the largest value and which half of the numbers will produce the smallest value. $\frac{n}{2}$

Larger $\boxed{L[0 \cdots \frac{n}{2}]}$

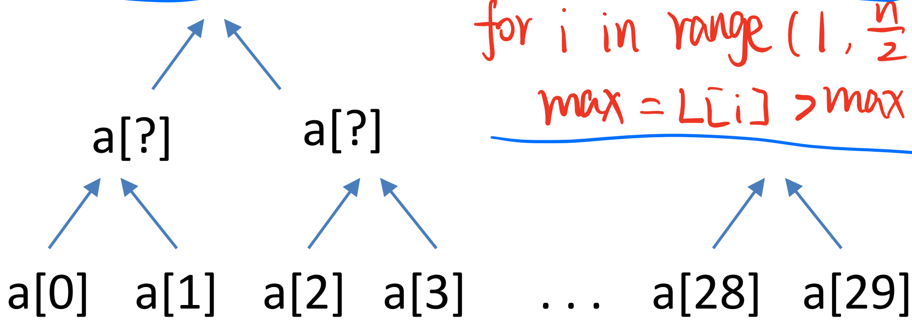
Smaller $\boxed{S[0 \cdots \frac{n}{2}]}$

```
for (i=0; i<n; i+=2)
    L[i/2], S[i/2] = a[i] > a[i+1] ? (a[i], a[i+1]) : (a[i+1], a[i])
max = L[0]; min = S[0]
```

```
for i in range(1, n/2)        n/2 - 1
    max = L[i] > max ? L[i] : max
```

```
for i in range(1, n/2)        n/2 - 1
    min = S[i] > min ? S[i] : min
```

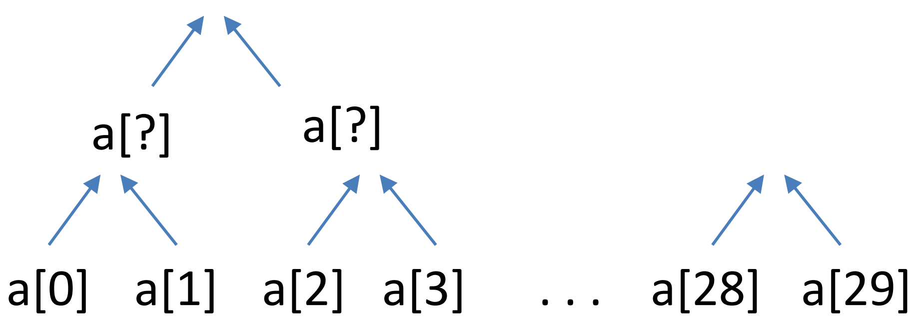
$\Rightarrow 1.5n - 2$



Program Complexity Exercise 1

- You are given 30 integers as the input. Write a procedure to output the largest value and the smallest integer value.
- Suppose a[0] to a[29] are storing these integers.

Do pairs of comparisons to decide which half of numbers will produce the largest value and which half of the numbers will produce the smallest value.



$n/2 + n/2-1 + n/2-1$

$=1.5n-2$

Supplementary materials for Analysis

- Why do we care about Big-Oh Analysis?
- Why can we write $n^2+3n+1=O(n^2)$? We know that $n^2+3n+1>n^2$!
- We said that $n^4=o(2^n)$, but if I set $n=3$, I have $3^4>2^3$, how does it come?

Supplementary materials for Analysis

- Why do we care about Big-Oh Analysis?

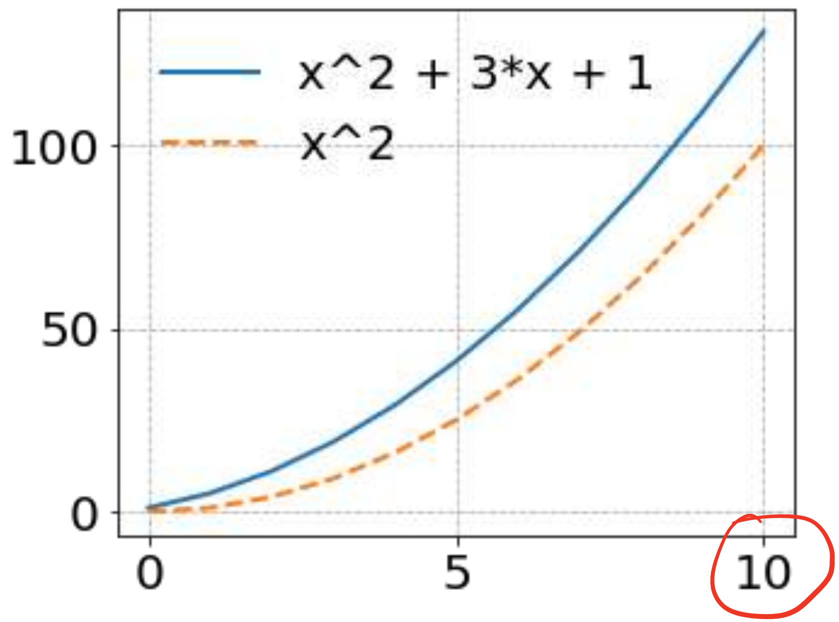
The running time we estimated for a program is usually the worst case analysis, i.e. the maximum time required by that program. Therefore, Analyzing the bound from above will make sense. This is why we usually do Big-Oh Analysis.

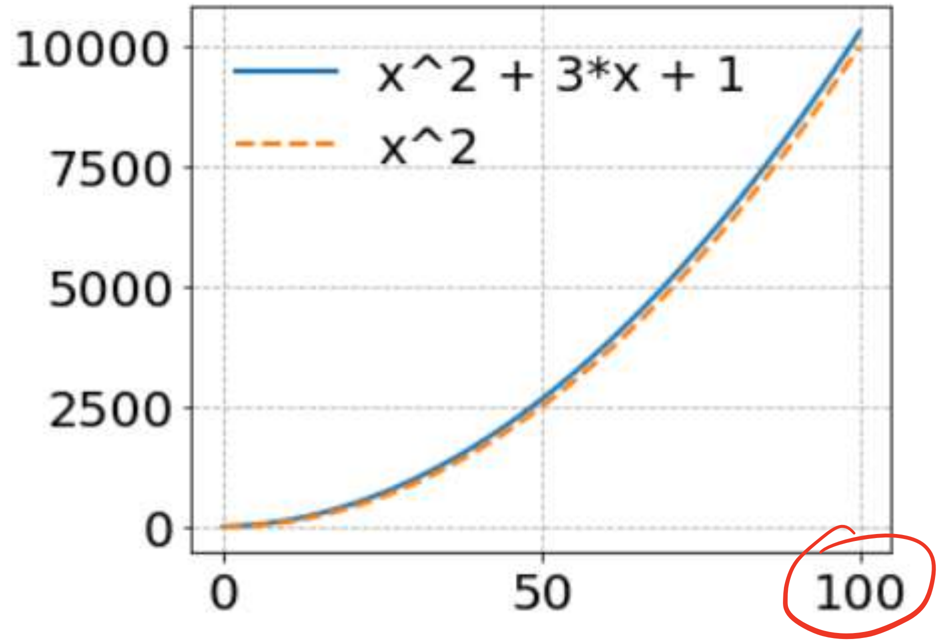
- Why can we write $n^2+3n+1=O(n^2)$? We know that $n^2+3n+1>n^2$!

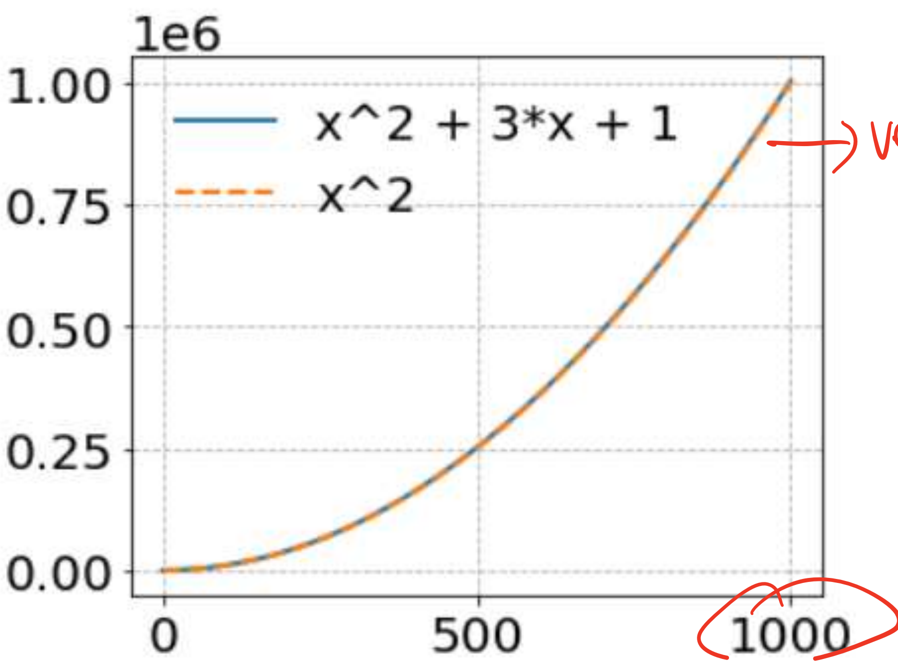
We have $n^2+3n+1 \le 5n^2$ for all n. We know that for any positive value c, $cn^2=O(n^2)$ according to the definition. Since the value of n^2+3n+1 is bounded by $5n^2$, we can also write $n^2+3n+1=O(n^2)$. Remember that O() represents a class of functions.

$\Rightarrow$ figure

Supplementary materials for Analysis







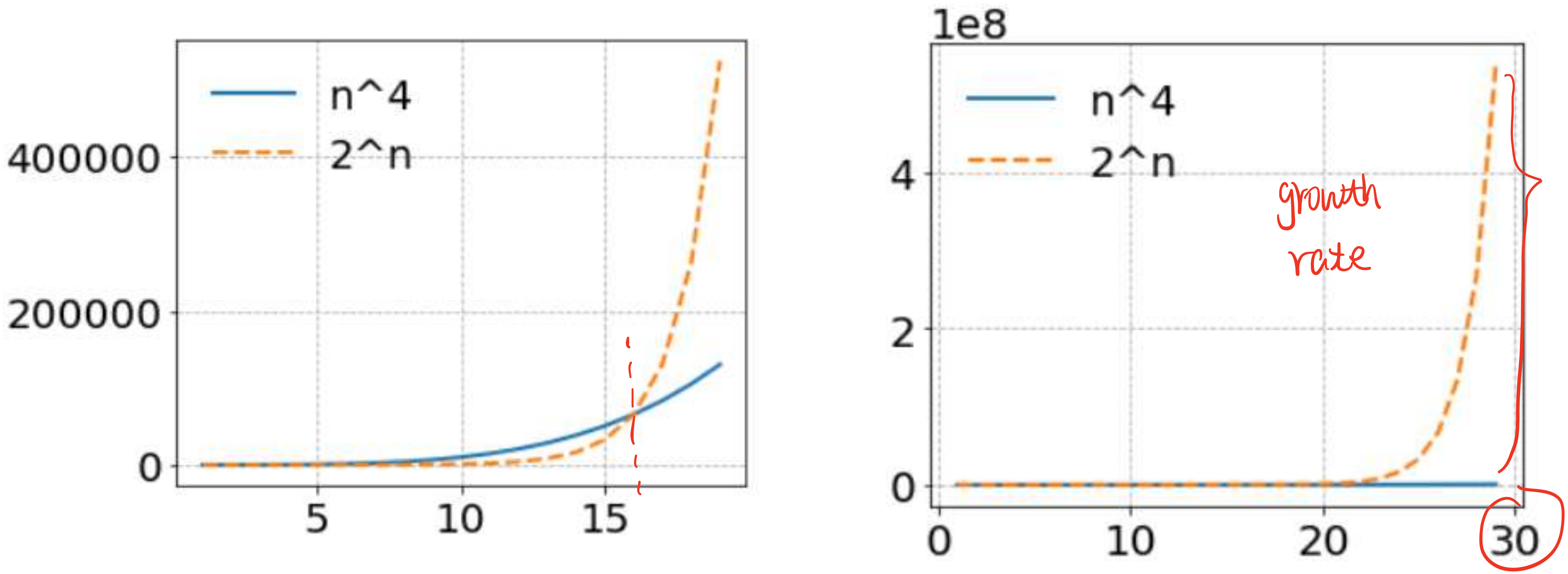
very close

Supplementary materials for Analysis

- We said that $n^4=o(2^n)$, but if I set $n=3$, I have $3^4>2^3$, how does it come?

All the asymptotic notations compare function values for sufficiently large input size. In this example, when $n>20$, we can know that $n^4<2^n$.

Supplementary materials for Analysis



Supplementary materials for Analysis

dominant term

- Suppose program 1 has worst case running time $T1(n)=3n^2+100n\log n+9$, program 2 has worst case running time $T2(n)=2^n+n$ and they can output the same results. Which program do you choose? Explain the reasons for the choice.
  Comparing the growth rates of these two functions, we can focus on the dominant terms.
  dominant term
  grow faster, choose T1
- Suppose program 1 has best case running time $T1(n)=3n^2$, program 2 has best case running time $T2(n)= n$ and they can output the same results. Which program do you choose? Explain the reasons for your answer.

The best-case running time is not always significant factor in choosing a program.

The bests-case scenarios rarely represent real-world situations,

The worst-case or average-case scenarios are usually more relevant.

Exercise 2

Analyze the running time of the following program

(a)
```
Void function1(int n)
{
        int i,j;
        int x=0;
        for(i=0;i<n;i++)
        {
                If(x<100)
                        x++;
                for(j=0;j<n;j++)
                        x+=j;
        }
}
```

O(n)

constant

O(n)

O(n²)

(b)
```
Void function2 (int n)
{
        int i,j;
        int x=200;
        for(i=0;i<n;i++)
        {
                If(x>100)
                        x--;
                else
                        for(j=0;j<n;j++)
                                x+=j;
        }
}
```

O(n²)

Exercise 2

Analyze the running time of the following program

(c)

```
void function3(int n) {
   if (n==1)
          return;
   for (int i=1; i<=n; i++) {
     for (int j=1; j<=n; j++)  {
        print("*");
        break;
     }
   }
}
```

Constant

O(n)

(d)

```
void function4(int n, int choice) {
   int x = 0;
   if (choice==1) {
     for (int i=1; i<=n; i++)
        for (int j=1; j<=20; j++)
          for (int k=n; k>1; k=k/2)
               x+=5;
   }
   else {
        for (int j=1; j<=n*n; j++)
          for (int k=0; k<n; k++)
               x+=3;
   }
}
```

$O(n \log n)$

→ n

→ constant

→ $\log_2 n$

→ n^2

→ n

$O(n^3)$

819 Josephus Problem

Description

n individuals labeled from 1 to *n* form a circle, which means the next person of the *n*-th person is the first person. Counting begins at the first person, and the *m*-th person counted will go out.

Then the counting restart at the next person of the one who went out, and still the m-th person counted will go out. Repeat the counting until all of the people have gone out.

In this problem, given *n* and *m*, please show the order they went out.

Input

A single line containing two integers *n* and *m* separated by a space.

Output

Print *n* integers in a single line denoting the labels of these *n* persons and indicating the order they went out. Please separate each two of these integers by a single space.

819 Josephus Problem

Sample Input	Sample Output
10 3	3 6 9 2 7 1 8 5 10 4 $\cancel{\square}$

No space symbol

1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10

1 → 2 → 4 → 5 → 6 → 7 → 8 → 9 → 10

1 → 2 → 4 → 5 → 7 → 8 → 9 → 10

1 → 2 → 4 → 5 → 7 → 8 → 10

Constraint

1 <= n, m <= 10^4

O(nm) algorithm can pass through all test cases

Which operation is needed for counting which person in a circle will go out?

1. Linked list
2. remove node

Array? Be careful the time complexity!

$d = m \% n$

819 Josephus Problem

Sample Input	Sample Output
10 3	3 6 9 2 7 1 8 5 10 4

Constraint
1 <= *n*, *m* <= 10^4

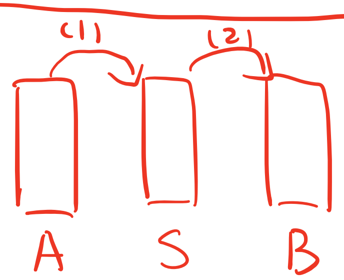
O(*nm*) algorithm can pass through all test cases

Which operation is needed for counting which person in a circle will go out?
%

744 Stack Shuffling

Description

After learning stack--the widely used data structure, Icy plan to play with it. The rule is as follows: There are 3 stacks A, B and S, where stack B and S are empty initially.



There are only two kinds of movement.

(1)Top element of A can only be moved onto the top of S.

(2)Top element of S can only be moved onto the top of B.

By repeating (1) and (2) until A and S are empty, all elements in A will be moved to B and the elements in B are permuted (the order may not be the same). So here comes the problem.

Given the initial order of elements in stack A and a final order of elements in B, can you cleverly judge whether the final order is possible to achieve?

744 Stack Shuffling

Input/Output

The input contains **multiple** test cases.

- The first line of input is an integer T (1 <= T <= 10) representing the number of test cases.
- For each test case, the first line gives an integer n ($1 \le n \le 3000$) indicating the number of elements in stack A,
- the following line gives n integers representing the corresponding elements in stack A (first element is bottom, last element is top and we guarantee that all the elements are distinct).
- Then, the next line contains an integer m (m <= 200) telling you how many permutations you have to judge
- and in each of the following m lines, there are n integers indicating the desired elements to be tested in stack B.

If the permutation is possible to achieve, print "Aye",
otherwise print "Impossible" in a separate line.

744 Stack Shuffling

Sample Input	Sample Output
1 T 5 n No. of element 1 2 3 4 5 Stack A 3 1 2 3 4 5 1 5 4 2 3 3 2 1 4 5 Stack B	Aye Impossible Aye

$S = \phi$ idxB = 0

while A is not empty
A → S
while Top of S is B[idxB]
S → B
check A.empty(), S.empty()

A
S 5 4 3 2
B 1

A is empty, finish. ⇒ 1 2 3 4 5 ⇒ Aye

A
S 5 4 3 2
B 1

The 2^{nd} element in B must be 2 ⇒ Impossible

A 1 2
S 5 4
B 3

A 1
S 5 4
B 3 2

A
S 5 4
B 3 2 1

⇒ A is empty ⇒ 3 2 1 4 5 ⇒ Aye

744 Stack Shuffling

Sample Input	Sample Output
1 5 1 2 3 4 5 3 1 2 3 4 5 1 5 4 2 3 3 2 1 4 5	Aye Impossible Aye

744 Stack Shuffling

For the last permutation "3 2 1 4 5", it can be achieved by the following operations:

A→**S**:5

A→**S**:4

A→**S**:3

S→**B**:3

A→**S**:2

S→**B**:2

A→**S**:1

S→**B**:1

S→**B**:4

S→**B**:5

Finally, the stack **B** will be "3 2 1 4 5".