

# EECS 3101

Prof. Andy Mirzaian



Computer Science  
and Engineering

120 Campus Walk

Assignment Project Exam Help  
**Iteration**  
<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

&

# Recursion

# **STUDY MATERIAL:**

- **[CLRS]** chapters 2, 4.1-2, 12.1, 31.1-2, 33.4
- **Lecture Note 4**

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

# TOPICS

- The algorithm design process:

- Central Tools: Iteration & Recursion
- Partial correctness: Assertions
- Termination: Measure of Progress

Assignment Project Exam Help

- Iterative Al

- Loop Inva <https://eduassistpro.github.io/>
- Incremental Method  
Add WeChat edu\_assist\_pro

- Recursive Algorithms:

- Recursive Pre- & Post- Condition & strong induction
- Divide-&-Conquer Method

# The Algorithm Design Process

We are given the pleasurable task of designing an algorithm  
for a computational problem.

- Where should we start? <https://eduassistpro.github.io/>
- What design pattern should we use?
- Are we on the right track? <https://eduassistpro.github.io/>
- Is there a systematic way to convince our colleagues, our friends, and all skeptics, that our algorithm is correct?
- Is our algorithm guaranteed to terminate on every possible input?
- Is there a better algorithm?
- ...

# Assertions

- An **assertion** is a true/false logical statement about the state of the variables and data structures of the algorithm.
- Although they appear as inserted comments, **assertions** are **not** descriptions of the code's intended purpose or an explanation of its action.
- Important examples of assertions are:
  - Pre-Condition
  - Post-Condition
  - Loop Invariant
- **Assignment Project Exam Help**  
<https://eduassistpro.github.io/>
- **Add WeChat edu\_assist\_pro**
- **What are assertions good for?**
  - They **guide us** along the design process. They should be with us from the initial design stage; to the middle, keeping us on track; to the end, convincing us that we have a **correct algorithm**, if we maintain correct assertions.
  - The design process might go through a **revise-&-repeat cycle**. If our assertions show us we are not quite on the right track, they usually also show us what is lacking and **suggest ways of correction!**

## Algorithm GuessWhat(n)

Pre-Cond: input n is a natural number

Post-Cond: output is ----- (fill in the blanks)

```
i ← 0  
j ← 1  
while i < n do  
    i ← i + 1  
    j ← j + j  
end-while  
return j  
end
```

§ What is true about values of i, j & n here?  
§ Any relationship between them?

Assignment Project Exam Help

## Algorithm GuessWhat

Pre-Cond: input n is a natural number

Post-Cond: output is  $2^n$  (how can we be sure?)

```
i ← 0  
j ← 1 } ← 1 =  $2^0$  & 0 ≤ n  
loop ← Loop-Invariant: j =  $2^i$  & i ≤ n  
    if i ≥ n then exit loop j =  $2^i$  & i < n  
    i ← i + 1 } 2j =  $2^{i+1}$  & i + 1 ≤ n  
    j ← j + j } so now: j' =  $2^{i'}$  & i' ≤ n  
end-loop j =  $2^i$  & i ≤ n & i ≥ n  
return j ←  
end
```

Termination:

Measure of Progress:

$$MP = n - i.$$

Each iteration reduces MP by 1.

Initial MP = n.

Loop exits when  $MP \leq 0$ .

# iterations = n.

} Establish LI  
(induction Base)

} (induction hypothesis)

} Maintain LI (induction)  
& progress  
} towards goal

} Exit loop, use LI  
& reach goal

## Algorithm ALG(Input)

Pre-Cond: Input satisfies problem instance specifications

ALG Code

Post-Cond: output satisfies problem solution requirements

end

Pre-Cond & ALG Code  $\rightarrow$  Post-Cond  
Assignment Project Exam Help

Pre-Cond

P

<https://eduassistpro.github.io/>

If input satisfies

Add WeChat `edu_assist_pro`

if ALG eventually

then the output is guaranteed to satisfy Post-Cond.

ALG Code

Post-Cond

Post-Cond is not guaranteed to hold

if input does not satisfy Pre-Cond, or

if ALG does not terminate on the given input.

Total Correctness = Partial Correctness + Termination.

Assertion0

⟨code fragment⟩

Assertion1

Assertion0 & ⟨code fragment⟩  $\Rightarrow$  Assertion1  
**Assignment Project Exam Help**

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

⟨code fragment⟩

Assertion1

## Sequential code:

Assertion 0

⟨code 1⟩

Assertion 1

⟨code 2⟩

.....

Assignment Project Exam Help

Assertion M-1

⟨code M⟩

Assertion M

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

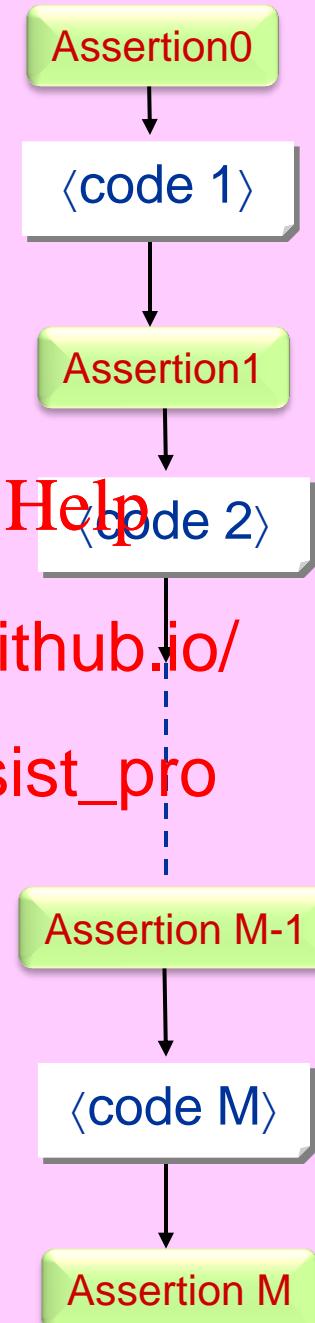
Assertion 0 & ⟨code 1⟩  $\Rightarrow$  Assertion 1

Assertion 1 & ⟨code 2⟩  $\Rightarrow$  Assertion 2

Assertion 2 & ⟨code 3⟩  $\Rightarrow$  Assertion 3

....

Assertion M-1 & ⟨code M⟩  $\Rightarrow$  Assertion M



## If-then-else:

Assertion0

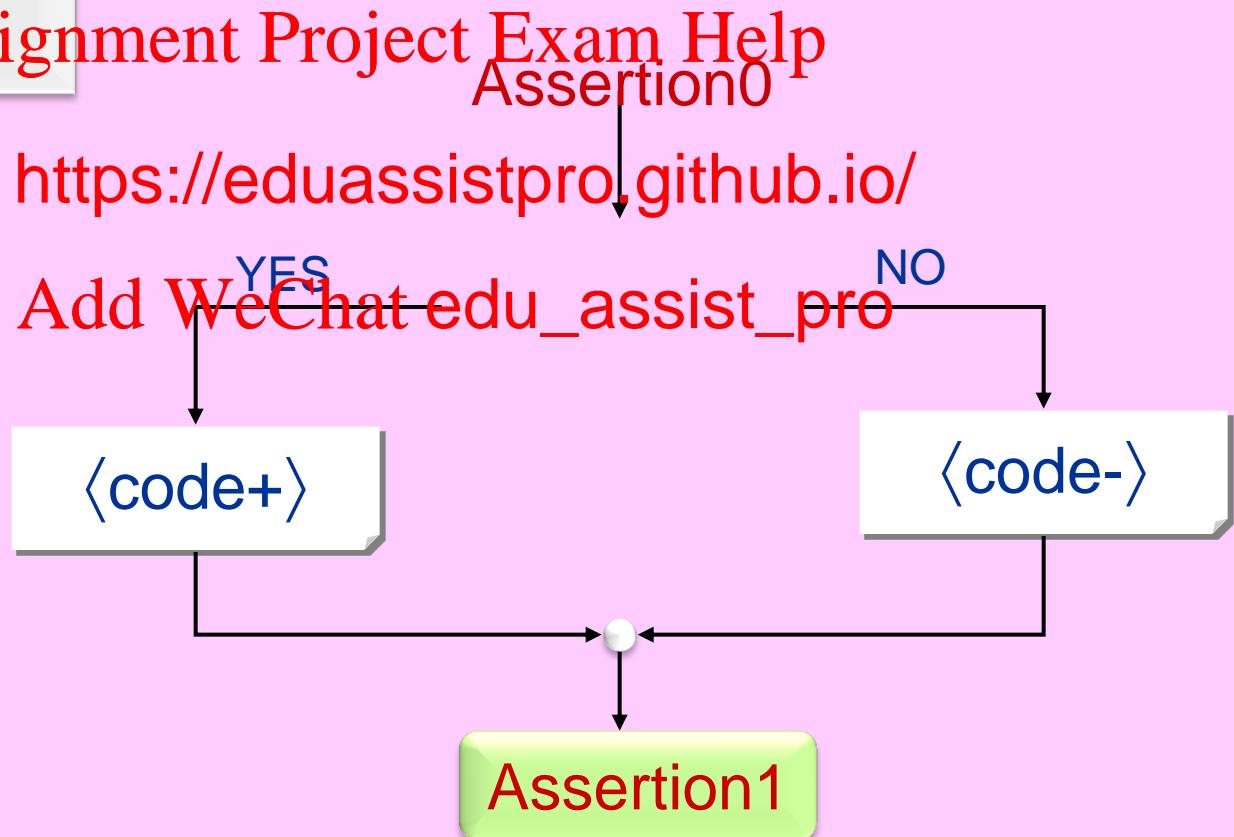
```
if <cond>
    then <code+>
    else <code->
```

Assertion1

$\text{Assertion0} \ \& \ \langle \text{cond} \rangle \ \& \ \langle \text{code+} \rangle \Rightarrow \text{Assertion1}$

and

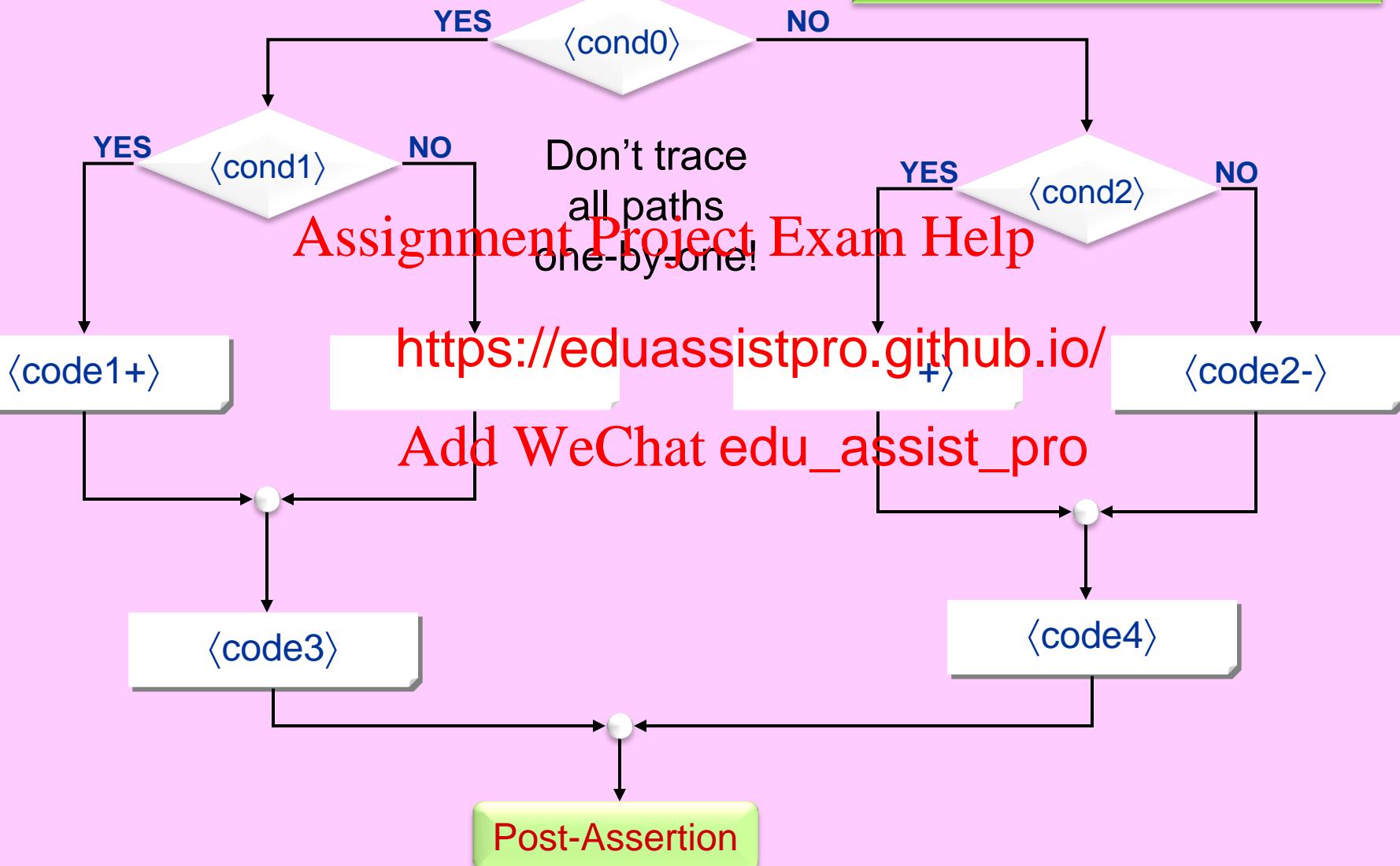
$\text{Assertion0} \ \& \ \neg \langle \text{cond} \rangle \ \& \ \langle \text{code-} \rangle \Rightarrow \text{Assertion1}$



Many ... many paths  
from Pre- to Post- Assertion.

Pre-Assertion

It suffices to verify each  
code fragment block only  
once!



# Loop Invariant

**Algorithm ALG(Input)**

Pre-Cond: Input satisfies problem instance specifications

Pre-Loop Code

**Loop:**

Loop Invariant

**if** <exit-condition> **then exit loop**

Loop Code

**end-loop**

Post-Loop

Post-Loop <https://eduassistpro.github.io/>

Post-Cond: output satisfies problem requirements

**end**

Add WeChat edu\_assist\_pro

Pre-Cond & Algorithm Code  $\Rightarrow$  Post-Cond

Loop-Invariant  $\leftarrow$  induction hypothesis

Pre-Cond & Pre-Loop-code  $\Rightarrow$  Loop-Invariant  $\leftarrow$  induction base

Loop-Invariant &  $\neg$ (exit-cond) & Loop-code  $\Rightarrow$  Loop-Invariant  $\leftarrow$  induction

Loop-Invariant & (exit-cond)  $\Rightarrow$  Post-Loop-Cond

Post-Loop-Cond & Post-Loop-code  $\Rightarrow$  Post-Cond  $\leftarrow$  clean up loose ends

ACTIONS ASSERTIONS

Assignment Project Exam Help

# ALGORITHM ALG (Input)

Pre-Cond: Input satisfies problem instance specifications

Pre-Loop Code

Pre-Cond &  
Pre-Loop-code  
 $\Rightarrow$  Loop-Invariant

Loop-Invariant &  
 $\neg \langle \text{exit-cond} \rangle$  & Loop-code  
 $\Rightarrow$  Loop-Invariant

Assignment Project Exam Help  
Loop Invariant

Loop-Invariant &  
 $\langle \text{exit-cond} \rangle \Rightarrow$   
Post-Loop-Cond

<https://eduassistpro.github.io/>

Add WeChat `edu_assist_pro`

$\langle \text{exit-cond} \rangle$

YES

Post-Loop-Cond

Post-Loop-Cond  
& Post-Loop-code  
 $\Rightarrow$  Post-Cond

Post-Loop Code

Post-Cond: output satisfies problem solution requirements

Pre-Cond : input A[1..n] is an array of numbers,  $n \geq 0$

EXAMPLE PROBLEM:  
Sum of array items.

Pre-Loop Code

Incremental  
Method

Loop Invariant

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat YES edu\_assist\_pro

Post-Loop-Cond

Post-Loop Code

Post-Cond: output is sum of the items in the input array

Start with the  
star of the show:

the Loop  
Invariant.

Pre-Cond : input A[1..n] is an array of numbers,  $n \geq 0$

Pre-Loop Code

Incremental  
Method

Loop Invariant:  $S = \text{sum}\{A[1..t]\}$

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat YES edu\_assist\_pro

Post-Loop-Cond

Post-Loop Code



Post-Cond: output is sum of the items in the input array

Pre-Cond : input A[1..n] is an array of numbers,  $n \geq 0$

Pre-Loop Code

establish LI

Loop Invariant:  $S = \text{sum}\{A[1..t]\}$

Assignment Project Exam Help

Incremental Method

<https://eduassistpro.github.io/>

Add WeChat YES **edu\_assist\_pro**

$S = \text{sum}\{A[1..n]\}$

**return S**

Post-Loop-Cond &  
Post-Loop-code  
 $\Rightarrow$  Post-Cond

Post-Cond: output is sum of the items in the input array

Pre-Cond : input A[1..n] is an array of numbers,  $n \geq 0$

Pre-Cond &  
Pre-Loop-code  
 $\Rightarrow$  Loop-Invariant

Incremental  
Method

$S \leftarrow 0; t \leftarrow 0$

Loop Invariant:  $S = \text{sum}\{A[1..t]\}$

Assignment Project Exam Help

Let's not use " $t = n$ ".  
Loop won't terminate with  
bad input, e.g.,  $n < 0$ .

<https://eduassistpro.github.io/>

Add WeChat yes edu\_assist\_pro

$S = \text{sum}\{A[1..n]\}$

**return S**

Post-Loop-Cond &  
Post-Loop-code  
 $\Rightarrow$  Post-Cond

Post-Cond: output is sum of the items in the input array

Pre-Cond : input A[1..n] is an array of numbers,  $n \geq 0$

Pre-Cond &  
Pre-Loop-code  
 $\Rightarrow$  Loop-Invariant

$S \leftarrow 0; t \leftarrow 0$

Incremental  
Method

Loop Invariant:  $S = \text{sum}\{A[1..t]\}$

Assignment Project Exam Help

Let's not use " $t = n$ ".  
Loop won't terminate with  
bad input, e.g.,  $n < 0$ .

<https://eduassistpro.github.io/>

Add WeChat  edu\_assist\_pro

$S = \text{sum}\{A[1..n]\}$

Post-Loop-Cond &  
Post-Loop-code  
 $\Rightarrow$  Post-Cond

**return S**

maintain LI,  
& progress

Post-Cond: output is sum of the items in the input array

Pre-Cond : input A[1..n] is an array of numbers,  $n \geq 0$

Pre-Cond &  
Pre-Loop-code  
 $\Rightarrow$  Loop-Invariant

Incremental  
Method

$S \leftarrow 0; t \leftarrow 0$

Loop Invariant:  $S = \text{sum}\{A[1..t]\}$

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat  edu\_assist\_pro

$S = \text{sum}\{A[1..n]\}$

Loop-Invariant &  
 $\neg(\text{exit-cond})$  &  
Loop-code  
 $\Rightarrow$  Loop-Invariant

**return S**

Post-Loop-Cond &  
Post-Loop-code  
 $\Rightarrow$  Post-Cond

Post-Cond: output is sum of the items in the input array

Pre-Cond : input A[1..n] is an array of numbers,  $n \geq 0$

Pre-Cond &  
Pre-Loop-code  
 $\Rightarrow$  Loop-Invariant

$S \leftarrow 0; t \leftarrow 0$

Incremental  
Method

Loop Invariant:  $S = \text{sum}\{A[1..t]\}$

Assignment Project Exam Help

Strengthen  
LI

Loop-Invariant &  
 $\langle \text{exit-cond} \rangle \not\Rightarrow$

$t+1$   
 $S + A[t]$

Add WeChat  $\text{edu\_assist\_pro}$

Post-Loop-Cond

Post-Loop-Cond &  
Post-Loop-code  
 $\Rightarrow$  Post-Cond

$S = \text{sum}\{A[1..n]\}$

Loop-Invariant &  
 $\neg \langle \text{exit-cond} \rangle \&$   
Loop-code  
 $\Rightarrow$  Loop-Invariant

return S

Post-Cond: output is sum of the items in the input array

Pre-Cond : input A[1..n] is an array of numbers,  $n \geq 0$

Pre-Cond &  
Pre-Loop-code  
 $\Rightarrow$  Loop-Invariant

$S \leftarrow 0; t \leftarrow 0$

Loop Invariant:  $S = \text{sum}\{A[1..t]\}$

Assignment Project Exam Help

Loop-Invariant &  
 $\langle \text{exit-cond} \rangle \Rightarrow$

Post-Loop-Cond

Post-Loop-Cond &  
Post-Loop-code  
 $\Rightarrow$  Post-Cond

$S = \text{sum}\{A[1..n]\}$

**return S**

Post-Cond: output is sum of the items in the input array

LI changed.  
Need to revise  
Pre-Loop Code  
or Loop Code ?

No revision  
needed.  
Lucky this time!

<https://eduassistpro.github.io/>

Add WeChat  edu\_assist\_pro

Loop-Invariant &  
 $\neg \langle \text{exit-cond} \rangle \&$   
Loop-code  
 $\Rightarrow$  Loop-Invariant

Termination:

Measure of Progress:  
 $MP = n - t$ .

Each iteration reduces MP by 1.

Initial  $MP = n$ .

Loop exits when  $MP \leq 0$ .

# iterations =  $\max\{n, 0\}$ .

Pre-Cond : input  $A[1..n]$  is an array of numbers,  $n \geq 0$

Pre-Cond &  
Pre-Loop-code  
 $\Rightarrow$  Loop-Invariant

$S \leftarrow 0; t \leftarrow 0$

Loop Invariant:  $S = \text{sum}\{A[1..t]\}$

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

$S = \text{sum}\{A[1..n]\}$

return  $S$

Loop-Invariant &  
 $\neg(\text{exit-cond}) \Rightarrow$

Post-Loop-Cond

Post-Loop-Cond &  
Post-Loop-code  
 $\Rightarrow$  Post-Cond

Loop-Invariant &  
 $\neg(\text{exit-cond}) \&$   
Loop-code  
 $\Rightarrow$  Loop-Invariant

Post-Cond: output is sum of the items in the input array

We certify this  
is a correct  
algorithm!

## Algorithm SUM(A[1..n])

Pre-Cond: input A[1..n] is an array of numbers,  $n \geq 0$

$S \leftarrow 0; t \leftarrow 0$

**Loop:**

Loop Invariant:  $S = \text{sum}\{A[1..t]\} \& t \leq n$

**if**  $t \geq n$  **then exit loop**

**end-loop**

$S = su$

**return** <https://eduassistpro.github.io/>  
e input array

Post-Cond:

**end**

Add WeChat edu\_assist\_pro

ACTIONS

ASSERTIONS

Pre-Cond & Algorithm Code  $\Rightarrow$  Post-Cond

Loop-Invariant  $\leftarrow$  induction hypothesis

Pre-Cond & Pre-Loop-code  $\Rightarrow$  Loop-Invariant  $\leftarrow$  induction base

Loop-Invariant &  $\neg(\text{exit-cond})$  & Loop-code  $\Rightarrow$  Loop-Invariant  $\leftarrow$  induction

Loop-Invariant &  $\langle \text{exit-cond} \rangle \Rightarrow$  Post-Loop-Cond

Post-Loop-Cond & Post-Loop-code  $\Rightarrow$  Post-Cond  $\leftarrow$  clean up loose ends

# Loop Invariant may need revision

If LI is **too strong**, it may require too much effort to either establish or maintain it.

Pre-Cond & Pre-Loop-code

Assignment  $\Rightarrow$  Loop-Invariant  
Project Exam Help

Loop-<https://eduassistpro.github.io/>  
Loop-co nt

Add WeChat edu\_assist\_pro  
Loop-Invariant &  
 $\Rightarrow$  Post-Loop-Cond

Post-Loop-Cond & Post-Loop-code

$\Rightarrow$  Post-Cond

If LI is **too weak**, it may not imply the induction, or may not imply the desired end goal.

# **INCREMENTAL**

Assignment Project Exam Help

**A**

**MS**

<https://eduassistpro.github.io/>

Add WeChat **edu\_assist\_pro**

**Incremental progress over time is an exponential multiplier.**

-Michael Erik

**Water droplets gathered incrementally bring about a sea.**

-Persian proverb

# TOPICS

- **Incremental Method & its Loop Invariant**
- **Problems:**
  - VLSI Chip Testing Project Exam Help
  - In-place <https://eduassistpro.github.io/>
  - Maximum Sum Subarray Add WeChat edu\_assist\_pro
  - Longest Smooth Sub-Array

Pre-Cond: S is a set of input objects

```
C ← ∅;  
Obtain Sol(∅)
```

Incremental  
Method

Loop Invariant:  $C \subseteq S$  &  $\text{Sol}(C)$  is solution to C

Assignment Project Exam Help

This is generic.  
It may need to be  
strengthened in a  
given application.

lected object from  $S - C$ ;

$C \leftarrow C \cup \{x\}$

YES  
Add WeChat edu\_assist\_pro

$C = S$  &  $\text{Sol}(C)$  is solution to C

On-Line:

Items are given to you one-by-one. You need to update solution before next object is given.  
Examples: SUM and on-line Insertion Sort.

return  $\text{Sol}(C)$

Off-Line:

You are given all input objects in advance of any computation. You may choose any previously unselected input object.  
Example: Selection Sort.

Post-Cond: output is solution to S

# Problem: VLSI Chip Testing

[CLRS Problem 4-5, page 109]

- Pre-Condition:  
Input is a set  $S$  of VLSI chips, each chip is either good or bad (unknown to us). Strictly more than half of these chips are good. (Note: this implies  $S \neq \emptyset$ .)
- Post-Condition: Output is one of the good chips from the input.

Assignment Project Exam Help

- Notation:  $MTHG(S)$  means “ $s \in S$  are Good”.  
<https://eduassistpro.github.io/>
- Primitive operation: Boolean  $\text{Test}(x,y)$  outputs “ $x$  and  $y$  are both good chips.”  
Add WeChat `edu_assist_pro`  
$$\text{Test}(x,y) = \begin{cases} \text{True} & \Rightarrow x \text{ & } y \text{ are either both good or both bad} \\ \text{False} & \Rightarrow \text{at least one of } x \text{ or } y \text{ is bad} \end{cases}$$

- Use this primitive operation to solve the problem.
- Comment: If good and bad chips are 50-50,  $\text{Test}$  would be unable to break the good-bad symmetry. If less than 50% are good, that's even worse!

# Where to start?

- Brute Force: Compare each chip against every other chip.  
This requires  $O(n^2)$  Tests.  
Can we do it faster, incrementally?
- Incremental method could be quite effective!      How?  
Suppose we pick a pair of chips x and y from S and invoke  $\text{Test}(x,y)$ .

## Assignment Project Exam Help

- If  $\text{Test}(x,y) = \text{False}$ ,  
then at least one of x  
We can discard both  
We still have  $\text{MTHG}(S)$ . So, S still conta
- If  $\text{Test}(x,y) = \text{True}$ , then what?  
If x & y both happen to be bad, then discarding both is still safe.  
But x & y could both be good. It's unsafe to discard them both.  
Why? You no longer have the guarantee that  $\text{MTHG}(S)$ !  
You may have inadvertently discarded all the good chips!
- How should we “repair” this shortcoming?  
Strengthen the Loop Invariant. .... P.T.O.

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

# Strengthen the Loop Invariant

- Notation:

$MTHG(S)$  = “More Than Half of the chips in  $S$  are Good”.

$AllGood(C)$  = “chips in set  $C$  are all good”.

$Uniform(C)$  = “chips in set  $C$  are either all good or all bad”.

Note:  $[MTHG(C) \ \& \ Uniform(C)] \Rightarrow [AllGood(C) \ \& \ C \neq \emptyset]$ .

## Assignment Project Exam Help

- Strengthen Loop Inv

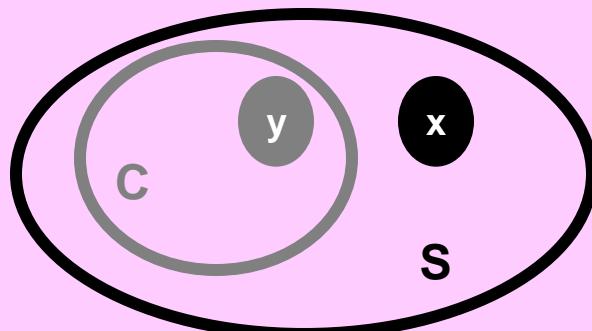
In addition to  $MTHG$ ( $C$ ) maintain a candidate <https://eduassistpro.github.io/> guarantee:  $Uniform(C)$ .

- Now, how can you maintain  $L$  and make

Idea: pick  $x \in S - C$  and  $y \in C$ , then  $\text{Test}(x,y)$ .

What happens?

What if  $S - C$  is empty? What if  $C$  is empty?



non-candidate  
tested chips  
“discarded”

Pre-Cond: input is a set  $S$  of  $n$  VLSI chips &  $MTHG(S)$ .

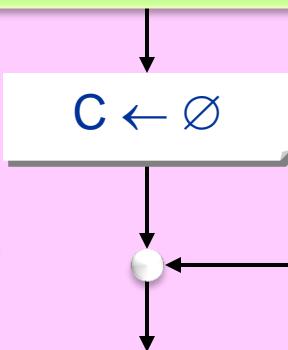
Pre-Cond &  
Pre-Loop-code  
 $\Rightarrow$  Loop-Invariant

Loop-Invariant &  
 $\langle$ exit-cond $\rangle \Rightarrow$   
Post-Loop-Cond

Post-Loop-Cond  
& Post-Loop-code  
 $\Rightarrow$  Post-Cond

return any one chip from  $C$

Post-Cond: output is a good chip from the input



Incremental  
Method

Loop Invariant:  $C \subseteq S$  & Uniform( $C$ ) &  $MTHG(S)$

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

$C \neq \emptyset$  & AllGood( $C$ )

arbitrary chip in  $S - C$

then  $C \leftarrow C \cup \{x\}$

n arbitrary chip in  $C$   
 $st(x,y)$

then  $C \leftarrow C \cup \{x\}$

else  $C \leftarrow C - \{y\};$   
 $S \leftarrow S - \{x,y\}$

end-do

Measure of Progress:  
 $MP = |S - C|$ .  
# iterations  $\leq n$ .

## Algorithm VLSIChipTesting( S ) § takes O(n) Tests

Pre-Cond: input is a set S of n VLSI chips & MTHG(S).

$C \leftarrow \emptyset$

**Loop:**

Loop Invariant:  $C \subseteq S$  & Uniform(C) & MTHG(S)

**if**  $C = S$  **then exit loop**

$x \leftarrow$  an arbitrary chip in  $S - C$

**if**  $C = \emptyset$

**then**  $C \leftarrow C \cup \{x\}$

**else do**

$y \leftarrow$  <https://eduassistpro.github.io/>

**if** T

**then**  $C \leftarrow C \cup \{x\}$

**else**  $C \leftarrow C - \{y\}$

**end-do**

**end-loop**

$C \neq \emptyset$  & AllGood(C)

**return** any one chip from C

Post-Cond: output is a good chip from the input

**end**

**Assignment Project Exam Help**

**Add WeChat** [edu\\_assist\\_pro](https://eduassistpro.github.io/)

# Problem: In-Place Tri-Partition

**INPUT:** An array A[1..n], where each item  $A[i] \in \{\text{red, green, blue}\}$ ,  $i=1..n$ .

**OUTPUT:** A permutation of A[1..n] so that all red items appear first, then all green items, then all blue items. Same colour items can appear in any order within their colour group.

**INPUT:**  1 2 3 4 5 6 7 8 9 10 11 12

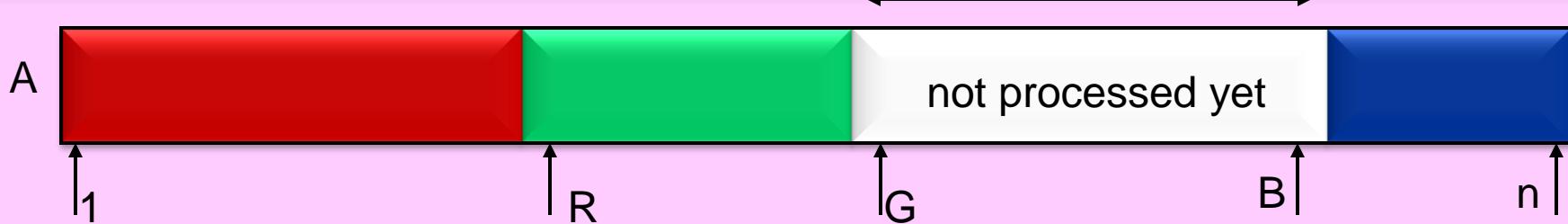
**OUTPUT:**  3 10 https://eduassistpro.github.io/ 6 1 9 12

**Restriction:** Add WeChat `edu_assist_pro`  
This has to be done **in-place**. We are allowed to use  
only O(1) additional scalar variables (e.g., array indices).  
[We **cannot** use O(n) additional memory cells such as lists.]

**Applications:** Partitioning in QuickSort & QuickSelect, In-place bucketing, etc.

**CLAIM:** We can do this partitioning in O(n) time **incrementally**.

# The Loop Invariant



**Loop Invariant:**

- $A[1..R-1]$  are all reds.
- $A[R..G-1]$  are all greens.
- $A[B+1..n]$  are all blues.
- $1 \leq R \leq G \leq B + 1$

**Maintain LI & Make Progress:**

Assignment Project Exam Help  
Reduce MP while maintaining LI? How?

<https://eduassistpro.github.io/>  
What's its colour?

**Measure of Progress:**

# unprocessed items.  
 $MP = B - G + 1$ .

Add WeChat edu\_assist\_pro

If     then     G++

If     A[G] = blue

then     A[G]  $\leftrightarrow$  A[B];     B--

**Exit Condition:**  $G > B$ .

(i.e., all items processed)

**Establish Loop Invariant:**

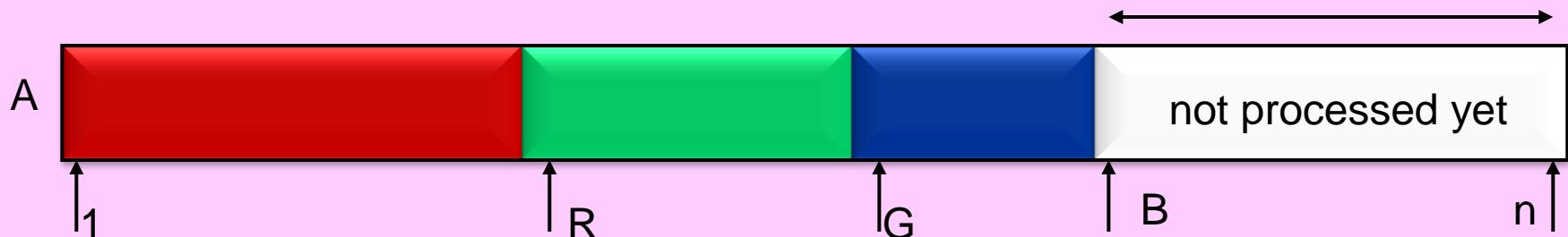
$(R, G, B) \leftarrow (1, 1, n)$ .

i.e., all  $n$  items are unprocessed.

If     A[G] = red

then     A[G]  $\leftrightarrow$  A[R];     R++;     G++

# Another Loop Invariant



Assignment Project Exam Help

**Exercise 1:**

Solve the 3-partition problem using the alternative  
Loop Invariant

Add WeChat `edu_assist_pro`

**Exercise 2:**

Generalize the 3-partition problem to 4-partition, 5-partition, ...

In general, for any integer  $C$ ,  $2 \leq C \leq n$ , show that the  $C$ -partition problem can be solved in  $O(Cn)$  time, using  $O(C)$  extra memory cells.

Therefore, if  $C$  is any constant, the  $C$ -partition problem can be solved in  $O(n)$  time, using  $O(1)$  extra memory cells.

# Problem: Maximum Sum Sub-Array

- **INPUT:** an array  $A[1..n]$  of arbitrary real numbers (positive, zero, negative).  
**OUTPUT:** a contiguous sub-array, say  $A[i+1..j]$ , of  $A[1..n]$  with max element sum (i.e.,  $A[i+1] + A[i+2] + \dots + A[j]$  is max possible).
- **Example:** [ 2, 1, -6, -3, 2, -4, 6, -2, -1, 1, 3, -4, 7, -5, 2, -3, 2, 1 ].
- **Brute Force Method**: Try every subarray  $A[i+1..j]$ , for all  $0 \leq i \leq j \leq n$ .  
This can be done in  $O(n^2)$ .

We can obtain  $\text{sum}(A)$  in  $O(n)$  time by using <https://eduassistpro.github.io/>. We can add  $O(n)$  pre-processing time to obtain  $\text{PrefixSum}[0..n]$ , where  $\text{PrefixSum}[t] = \text{sum}(A[1..t])$ , for  $t=0, 1, \dots, n$ . Add WeChat [edu\\_assist\\_pro](https://edu_assist_pro) incrementally):

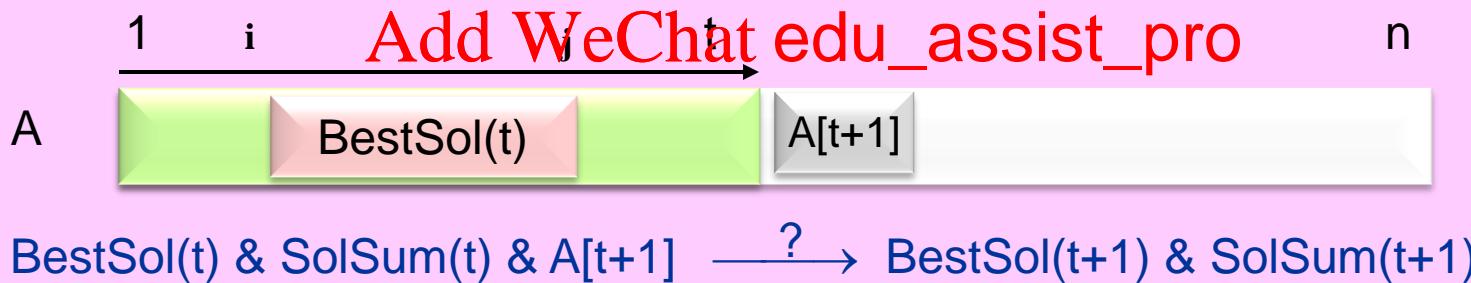
```
PrefixSum[0] ← 0;  for t ← 1..n do PrefixSum[t] ← PrefixSum[t-1]+A[t]
```

Now,  $\text{sum}(A[i+1..j]) = \text{PrefixSum}[j] - \text{PrefixSum}[i]$ .

- Can we solve the problem in sub-quadratic time incrementally?  
Let's try ..... P.T.O.

# Incremental Solution

- $A[1..t]$  = prefix of A considered incrementally so far.
- Suppose maximum sum subarray of  $A[1..t]$  is  $A[i+1..j]$ ,  $0 \leq i \leq j \leq t \leq n$ .  
Let's denote that solution by:  
 $\text{BestSol}(t) = A[i+1..j]$ , and  $\text{SolSum}(t) = \text{sum}(A[i+1..j])$ .
- **Loop Invariant:** ~~Assignment Project Exam Help~~  
Sol  
and <https://eduassistpro.github.io/>



Enough info in Loop Invariant?

# Examine the Loop Invariant

LI(t) & A[t+1]

BestSol(t) = A[i+1..j],  
SolSum(t) = sum(A[i+1..j]),  
A[t+1]

Maintain LI  
& progress

LI(t+1)

BestSol(t+1) = ?,  
SolSum(t+1) = ?

which  
case  
are  
we  
in?

**Case 1:** A[t+1] is not part of BestSol(t+1):

Assignment Project Exam Help

**Case 2:** A[t+1] <https://eduassistpro.github.io/>

BestSol(t+1) = a suffix

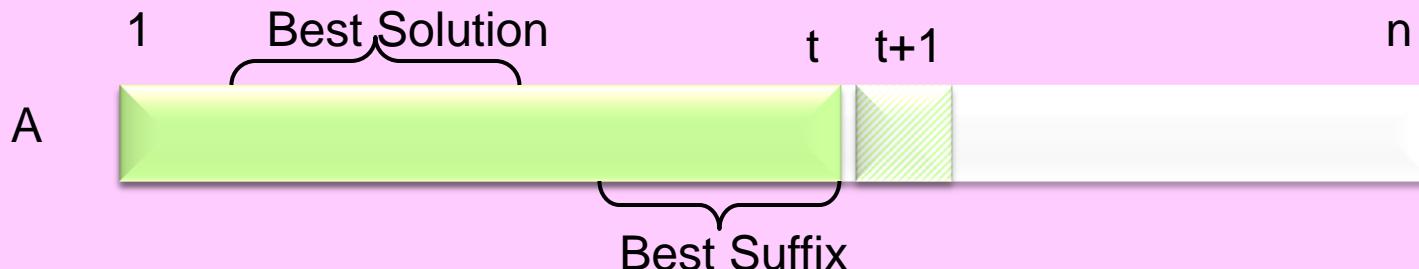
Which suffix?

Add WeChat edu\_assist\_pro

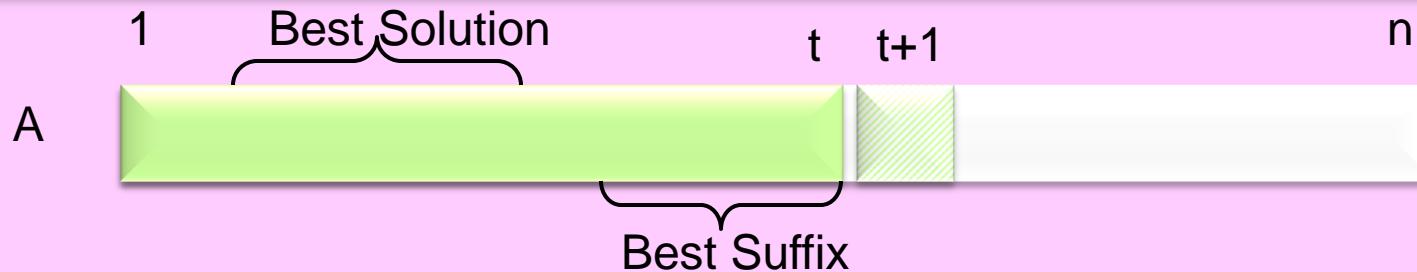
LI is not strong enough. It should

t is the

maximum sum suffix of A[1..t] and its element sum.



# Revise the Loop Invariant



**Loop Invariant:**  $\text{BestSol}(t) = A[i+1..j], \quad t \leq n, \quad \text{BestSuffix}(t) = A[k+1..t],$   
 $\text{SolSum}(t) = \sum(A[i+1..j]), \quad \text{SuffixSum}(t) = \sum(A[k+1..t])$

LI(t) & A[t+1]

<https://eduassistpro.github.io/>

SuffixSum(t+1) is better of two choices

(the one with bigger sum)

1) A[t+1] excluded:

$\text{BestSuffix}(t+1) = A[t+2..t+1]$  (i.e., nil)

$\text{SuffixSum}(t+1) = 0$

$\text{stSol}(0) \leftarrow A[1..0]$  (nil)  
 $\text{ISum}(0) \leftarrow 0$

$\text{BestSuffix}(0) \leftarrow A[1..0]$   
 $\text{SuffixSum}(0) \leftarrow 0$

2) A[t+1] included:

$\text{BestSuffix}(t+1) = (\text{BestSuffix}(t), A[t+1])$

$\text{SuffixSum}(t+1) = \text{SuffixSum}(t) + A[t+1]$

$\text{BestSol}(t+1) = \text{BestSol}(t) \text{ or } \text{BestSuffix}(t+1),$   
whichever has bigger sum.

Pre-Cond: input is an array  $A[1..n]$  of arbitrary integers.

Incremental  
Method

$(i, j, k, t) \leftarrow (0, 0, 0, 0)$   
 $SolSum \leftarrow SuffixSum \leftarrow 0$

Loop Invariant:

$BestSol(t) = A[i+1..t]$

$SolSum = \text{sum}(A[i+1..j])$

$BestSuffix(t) = A[k+1..t]$

$SuffixSum = \text{sum}(A[k+1..t])$

$t \geq n$

NO

YES

$BestSol(n) = A[i+1..j]$

**return**  $A[i+1..j]$

Add WeChat [edu\\_assist\\_pro](https://eduassistpro.github.io/)

$[t] > 0$   
 $t \leftarrow t+1$   
 $m \leftarrow 0; k \leftarrow t$   
**if**  $S$  **then**  $Sum$   
**then**  $SolSum \leftarrow SuffixSum;$   
 $(i, j) \leftarrow (k, t)$

Post-Cond: output is max-sum-subarray of input

$MP = n - t$ .  
 $\# \text{ iterations} \leq n$ .

## Algorithm MaxSumSubArray( A[1..n] ) § takes O(n) time

Pre-Cond: input is an array A[1..n] of arbitrary integers.

(i, j, k)  $\leftarrow$  (0, 0, 0) § t  $\leftarrow$  0

SolSum  $\leftarrow$  SuffixSum  $\leftarrow$  0

for t  $\leftarrow$  1 .. n do

Loop Invariant: BestSol(t) = A[i+1..j],

SolSum = sum(A[i+1..j]),

Assignment Project Exam Help  
Post-SuffixSum(A[1..t])

if SuffixSu

then Su

<https://eduassistpro.github.io/>

else SuffixSum  $\leftarrow$  0; k  $\leftarrow$

if SolSum  $<$  SuffixSum

then SolSum  $\leftarrow$  SuffixSum;

(i, j)  $\leftarrow$  (k, t)

end-for

BestSol(n) = A[i+1..j]

return A[i+1..j]

Post-Cond: output is max-sum-subarray of input

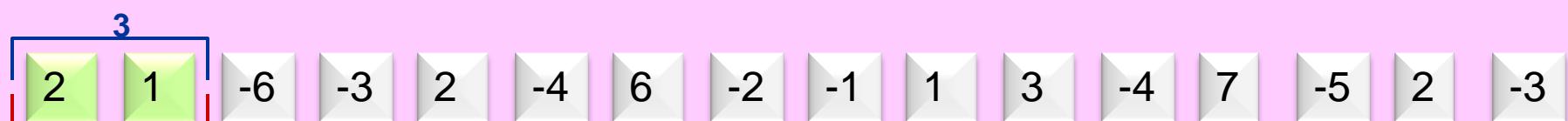
end

# Example run of the algorithm

BestSol



BestSuffix



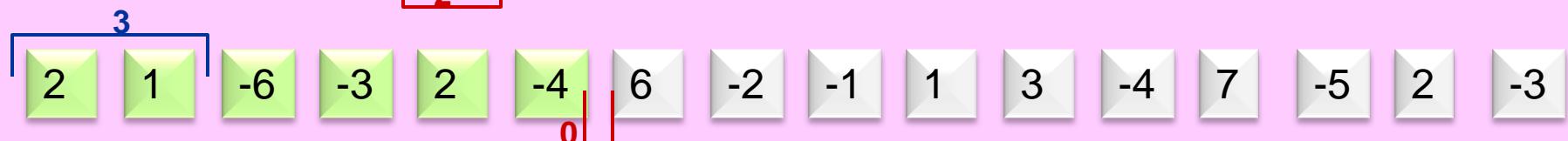
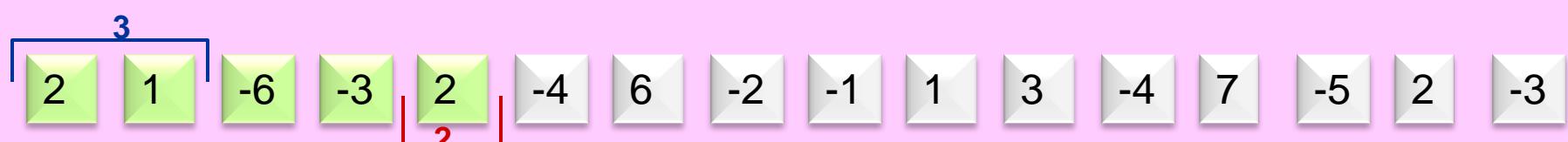
Assignment Project Exam Help



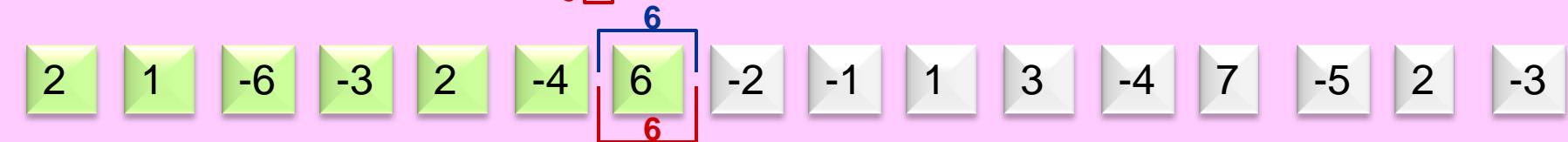
<https://eduassistpro.github.io/>



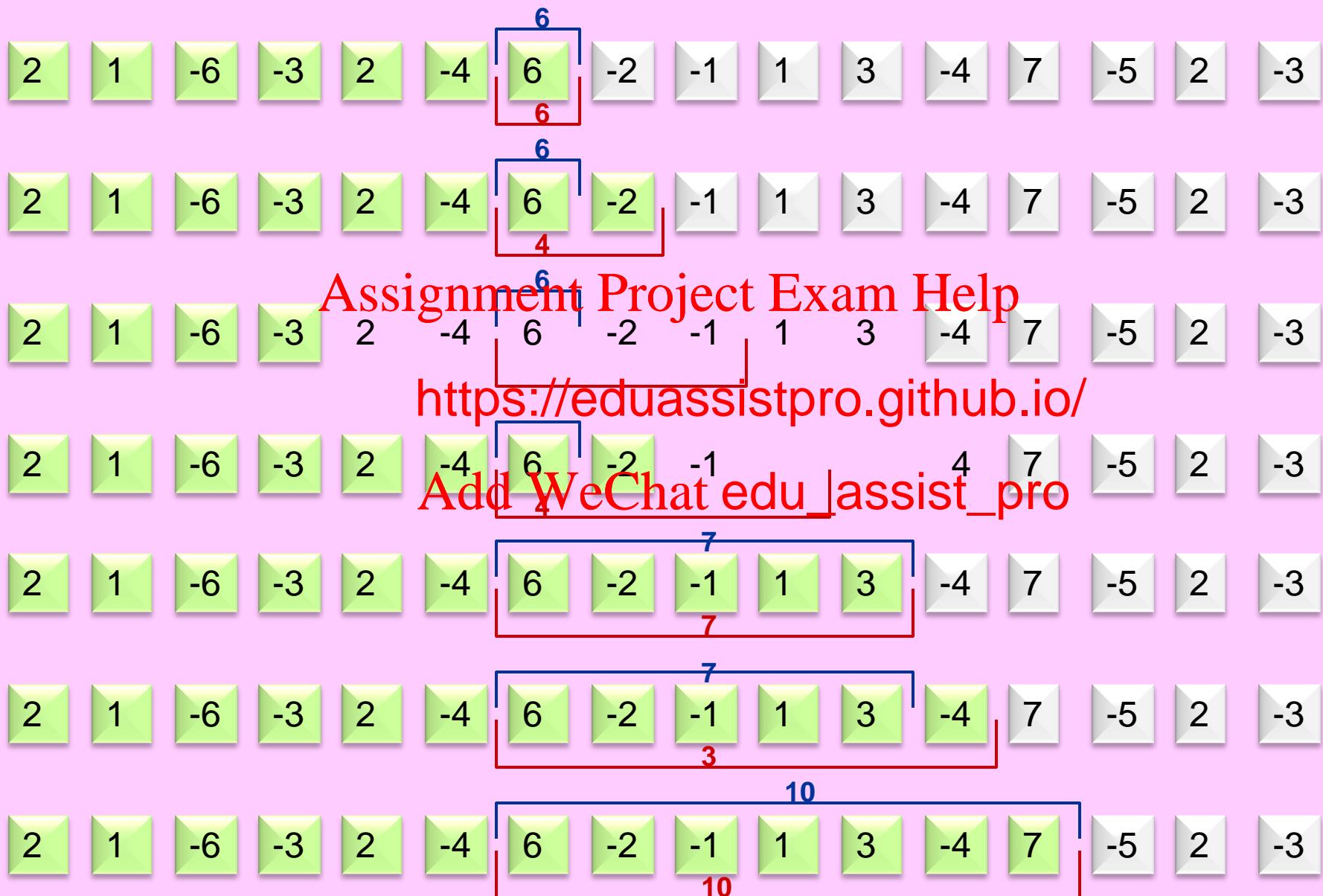
Add WeChat [edu\\_assist\\_pro](https://edu_assist_pro)



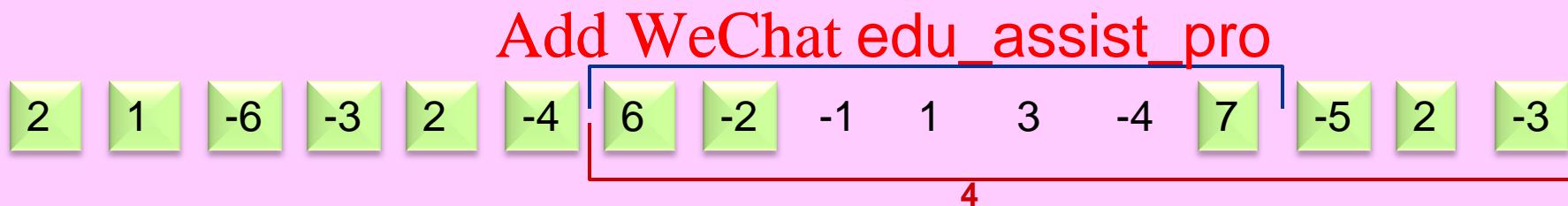
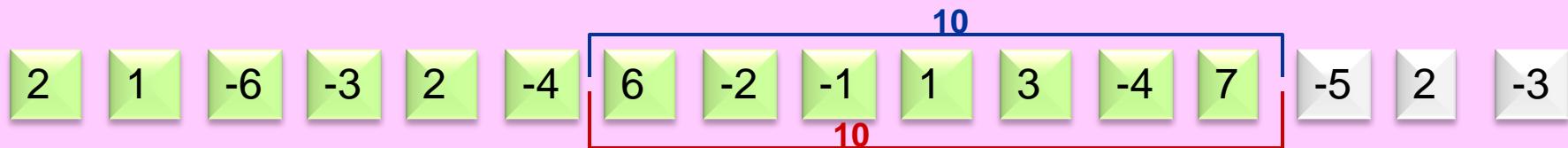
https://edu\_assist\_pro



# Example run of the algorithm



# Example run of the algorithm



Maximum Sum Sub-Array

# Problem: Longest Smooth Sub-Array

- Consider a real number  $\Delta > 0$ .

An array S of numbers is said to be  **$\Delta$ -smooth**, if no pair of items in S have a difference more than  $\Delta$ , i.e.,  $\max(S) - \min(S) \leq \Delta$ .

**Example:**  $S = [3, 4, 9, 5, 3, 6]$  is 7-smooth because  $\max(S) - \min(S) = 9-3 \leq 7$ .

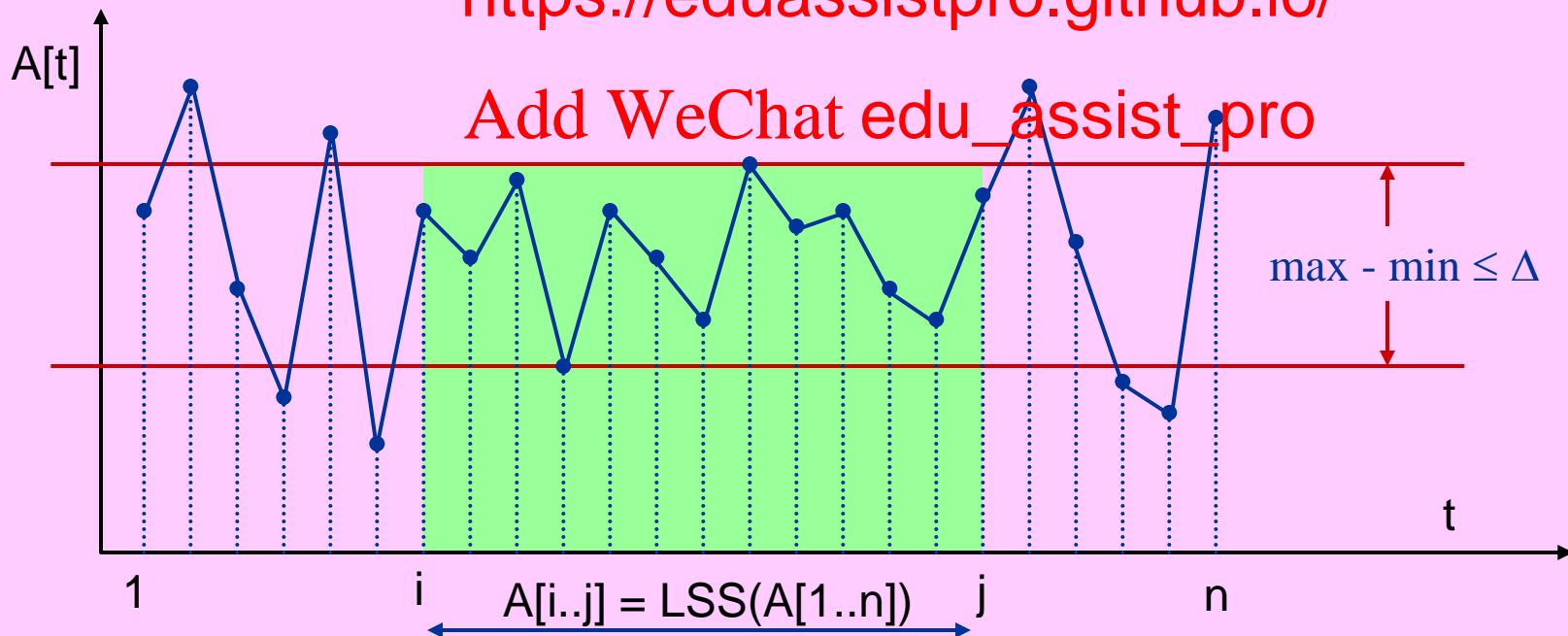
- Longest Smooth Subarray (LSS) Problem:**

**INPUT:** an array  $A[1..n]$  of numbers, and a number  $\Delta > 0$ .

**OUTPUT:** the longest smooth subarray of  $A[1..n]$ .

- Example:**  $\Delta = 8$  ,  $A$

<https://eduassistpro.github.io/>



# Where to start?

- **Brute Force Method:** Try every subarray  $A[i..j]$ , for all  $1 \leq i \leq j \leq n$ .  
This can be done in  $O(n^2)$  time. How?
- Can we solve the problem in sub-quadratic time?  
By the **Incremental Method**?
- Start with the obvious incremental LI and gradually strengthen it till it's right.

Assignment Project Exam Help

- LSS( $A[1..t]$ ) denotes <https://eduassistpro.github.io/>
- Our Loop Invariant so far includes:  
 $LI(a): A[i..j] = LSS(A[1..t]), i \leq j \leq t \leq$  Add WeChat edu\_assist\_pro
- How do we maintain  $LI(a)$  going from  $A[1..t-1]$  to  $A[1..t]$ ?  
Like MaxSumSubArray, we see we need the best suffix info too.
- LSX( $A[1..t]$ ) denotes the Longest  $\Delta$ -Smooth Suffix of  $A[1..t]$ .

# Strengthen the Incremental Loop Invariant

- LI(a):  $A[i..j] = LSS(A[1..t])$ ,  $i \leq j \leq t \leq n$ .
- $LSS(A[1..t]) = LSS(A[1..t-1])$  if  $A[t]$  is not part of  $LSS(A[1..t])$   
 $LSS(A[1..t]) = LSX(A[1..t])$  if  $A[t]$  is part of  $LSS(A[1..t])$

- Strengthen Loop Invariant  
LI(b):  $A[k..t] =$

- How do we maintain <https://eduassistpro.github.io/>

- Suppose  $LSX(A[1..t-1]) = A[k'..t-1]$ . This

Add  $A[t]$  to it to get  $A[k'..t]$ .

If  $A[k'..t]$  is  $\Delta$ -smooth, then  $LSX(A[1..t]) = A[k'..t]$ . Why?

What if  $A[k'..t]$  is not  $\Delta$ -smooth?

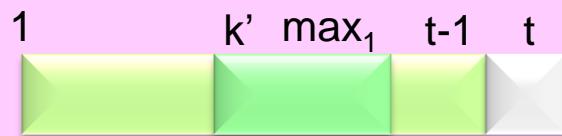
P.T.O.

# Further Strengthen the Loop Invariant

LI(a):  $A[i..j] = LSS(A[1..t])$ ,  $i \leq j \leq t \leq n$ .

LI(b):  $A[k..t] = LSX(A[1..t])$ ,  $k \leq t$ .

- $A[min_1] =$  rightmost minimum of  $A[k' .. t-1]$  &  
 $A[max_1] =$  rightmost maximum of  $A[k' .. t-1]$ .



## Assignment Project Exam Help

- $A[k'..t-1]$  is  $\Delta$ -smooth

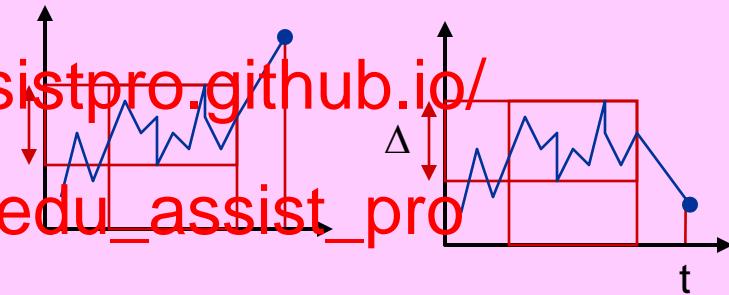
So, either

$$\text{or } A[t] - \Delta < A[min_1]$$

but not both.

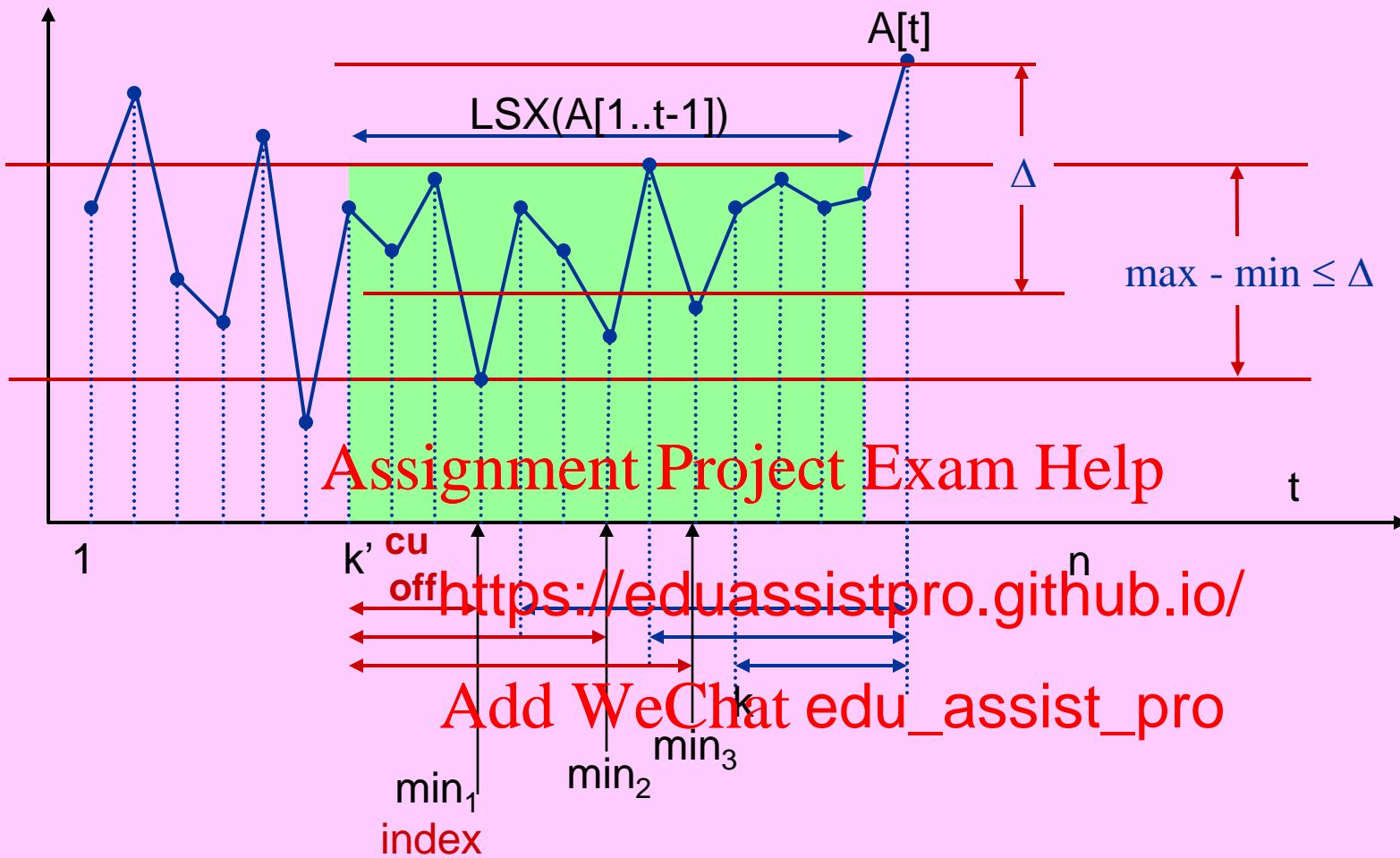
<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro



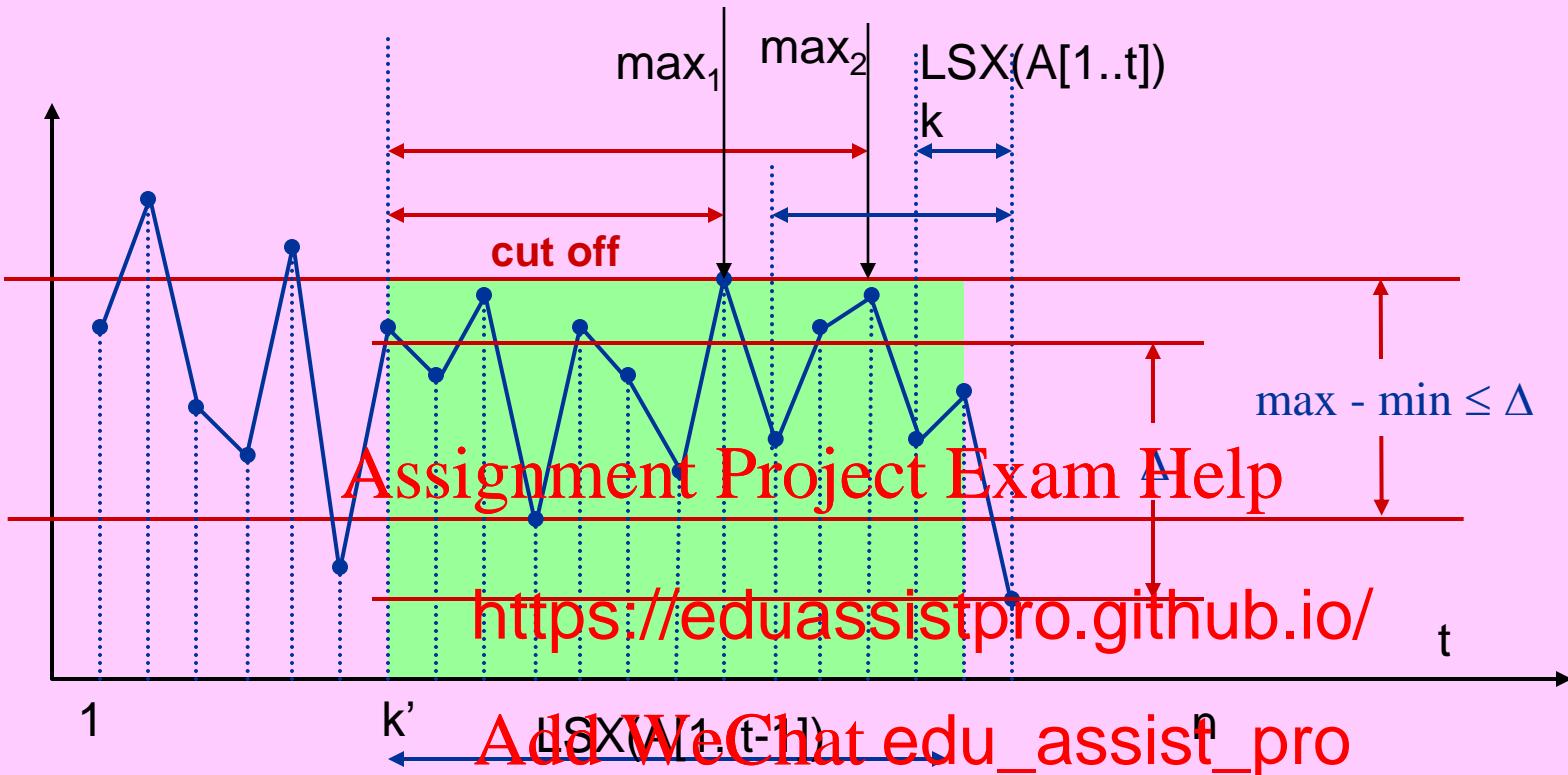
- If  $A[t] - \Delta > A[min_1]$ , then no part of  $A[k'.. \min_1]$  can be in  $LSX(A[1..t])$ .  
If  $A[t] + \Delta < A[\max_1]$ , then no part of  $A[k'.. \max_1]$  can be in  $LSX(A[1..t])$ .
- To get to  $k$ , we upgrade  $k'$  to  $1+ \min_1$  or  $1+\max_1$ , whichever is appropriate.

But we may not be done yet! ..... P.T.O.



```

 $m \leftarrow 1$ 
while  $A[t] - \Delta > A[\min_m]$  do  $k \leftarrow 1 + \min_m$ ;  $m++$ 
  
```



```

 $m \leftarrow 1$ 
while  $A[t] + \Delta < A[\max_m]$  do  $k \leftarrow 1 + \max_m$ ;  $m++$ 

```

# Strengthen Loop Invariant

- Strengthen LI to include these iterated min/max info.
- Two lists: MinL and MaxL.

At the end of iteration  $t$  they are:

$$\text{MaxL} = \langle \max_1, \max_2, \dots, \max_q \rangle$$

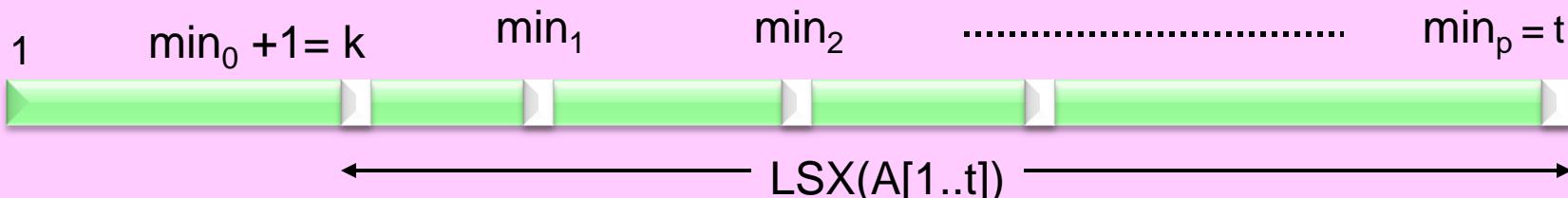
**Assignment Project Exam Help**  
 $k-1 = \max_0 < \max_1 < \max_2 < \dots \max_{q-1} < \max_q = t$ , and

$A[\max_m] = \text{rightmost minimum of } A[\min_{m-1}+1..t]$ , for  $m = 1..q$

$$\text{MinL} = \langle \min_1, \min_2, \dots, \min_p \rangle$$

$k-1 = \min_0 < \min_1 < \min_2 < \dots \min_{p-1} < \min_p = t$ , and

$A[\min_m] = \text{rightmost minimum of } A[\min_{m-1}+1..t]$ , for  $m = 1..p$ .



# The final Loop Invariant

## LOOP INVARIANT:

LI(a):  $A[i..j] = LSS(A[1..t])$ ,  $i \leq j \leq t \leq n$ .

LI(b):  $A[k..t] = LSX(A[1..t])$ ,  $k \leq t$ .

LI(c): MinL and MaxL lists satisfy the specifications described below.

MinL =  $\langle min_1, min_2, \dots, min_p \rangle$

$k-1 = \min_0 < \min_1 < \min_2 < \dots < \min_p < \min_{p+1} = t$ , and  
 $A[\min_m] = \text{rightmost minimum of } A[\min_{m-1} + 1..t]$ , for  $m = 1..p$ .

MaxL =  $\langle max_1, m \rangle$

$k-1 = \max_0 < \max_1 < \max_2 < \dots < \max_q = t$ , and  
 $A[\max_m] = \text{rightmost maximum of } A[\min_{m-1} + 1..t]$ , for  $m = 1..q$ .

## Operations on lists MinL and MaxL:

- $\text{Rear}(L)$  = return rear item in list L (lowest index). Don't alter L.
- $\text{Front}(L)$  = return front item in list L (highest index, i.e., t). Don't alter L.
- $\text{PopRear}(L)$  = remove and return rear item of L.
- $\text{PopFront}(L)$  = remove and return front item of L.
- $\text{PushFront}(t,L)$  = push item t to front of list L.
- $\text{PushRear}$  not needed.

# Maintain LI( $b+c$ ) in iteration $t-1$ to $t$

1. MinUpgrade: While  $A[t] - \Delta > A[\min_m]$ , we need to upgrade lowest index in LSX.
2. MaxUpgrade: While  $A[t] + \Delta < A[\max_m]$ , we need to upgrade lowest index in LSX.
3. FrontCleanUp:  $t$  is part of the new suffix & must be pushed to the front of both lists.  
Before doing that, remove front items in MinL (MaxL) that would no longer be rightmost minima (maxima) after  $t$  is added.

## Procedure MinUpgrade

```
while MinL ≠ ∅ & 1 + PopRear(MinL)
  while MaxL ≠ ∅ & https://eduassistpro.github.io/ MaxL)
```

## Procedure MaxUpgrade

```
while MaxL ≠ ∅ & A[t] + Δ < A[Rear(MaxL)] 1 + PopRear(MaxL)
  while MinL ≠ ∅ & Rear(MinL) < k do PopRear(MinL)
```

## Procedure FrontCleanUp

```
while MinL ≠ ∅ & A[t] ≤ A[Front(MinL)] do PopFront(MinL)
  while MaxL ≠ ∅ & A[t] ≥ A[Front(MaxL)] do PopFront(MaxL)
  PushFront(t, MinL)
  PushFront(t, MaxL)
```

only 2 pushes

all Pops

Pre-Cond: input is an array  $A[1..n]$  of numbers &  $\Delta > 0$ .

Pre-Cond &  
Pre-Loop-code  
 $\Rightarrow$  Loop-Invariant

Incremental  
Method

$(i, j, k, t) \leftarrow (0, 0, 0, 0)$   
 $(\text{MinL}, \text{MaxL}) \leftarrow (\emptyset, \emptyset)$

### LOOP INVARIANT:

- LI(a):  $A[i..j] = \text{LSS}(A[1..t])$ ,  $i \leq t$ .  
LI(b):  $A[k..t] = \text{LSX}(A[1..t]), \quad k \leq t$ .  
LI(c): MinL and Max

Loop-Invariant &  
 $\neg(\text{exit-cond}) \&$  Loop-code  
 $\Rightarrow$  Loop-Invariant

$\text{LSS}(A[1..n]) = A[i..j]$

$t \geq n$

YES

Loop-Invariant &  
 $\langle \text{exit-cond} \rangle \Rightarrow$   
Post-Loop-Cond

Post-Loop-Cond  
& Post-Loop-code  
 $\Rightarrow$  Post-Cond

return  $A[i..j]$

### Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat [edu\\_assist\\_pro](#)

if  $j - i < t - k$   
then  $(i, j) \leftarrow (k, t)$

Nested loops.  
Quadratic  
time ???

$$MP = 3(n - t) + |\text{MinL}| + |\text{MaxL}|.$$

Total iterations in all loops  $\leq 3n$ .

Post-Cond: output is longest  $\Delta$ -smooth subarray of  $A[1..n]$

## Algorithm LongestSmoothSubArray( $A[1..n]$ , $\Delta$ )      § takes $O(n)$ time

Pre-Cond: input is an array  $A[1..n]$  of numbers and  $\Delta > 0$ .

$(i, j, k) \leftarrow (0, 0, 0); (\text{MinL}, \text{MaxL}) \leftarrow (\emptyset, \emptyset)$    §  $t \leftarrow 0$

**for**  $t \leftarrow 1 .. n$  **do**   **Loop Invariant:** LI(a):  $A[i..j] = \text{LSS}(A[1..t])$ ,  $i \leq j \leq t \leq n$ .

LI(b):  $A[k..t] = \text{LSX}(A[1..t])$ ,  $k \leq t$ .

LI(c): MinL and MaxL as described.

§ maintain LI(b) and LI(c):

§ MinUpgrade:

**while**  $\text{MinL} \neq \emptyset$  &  $A[t] - \Delta > A[\text{Rear}(\text{MinL})]$  **do**  $k \leftarrow 1 + \text{PopRear}(\text{MinL})$

**while**  $\text{MaxL} \neq \emptyset$  &  $A[t] < A[\text{Rear}(\text{MaxL}) + 1]$  **do**  $\text{PopRear}(\text{MaxL})$

§ MaxUpgrade

**while**  $\text{MaxL} \neq \emptyset$  **do**  $k \leftarrow 1 + \text{PopRear}(\text{MaxL})$

**while**  $\text{MinL} \neq \emptyset$  **do**  $k \leftarrow 1 + \text{PopRear}(\text{MinL})$

§ FrontCleanUp:

**while**  $\text{MinL} \neq \emptyset$  &  $A[t] \leq A[\text{Front}(\text{MinL})]$  **do**  $\text{PopFront}(\text{MinL})$

**while**  $\text{MaxL} \neq \emptyset$  &  $A[t] \geq A[\text{Front}(\text{MaxL})]$  **do**  $\text{PopFront}(\text{MaxL})$

$\text{PushFront}(t, \text{MinL}); \text{PushFront}(t, \text{MaxL})$

§ maintain LI(a):

**if**  $j - i < t - k$  **then**  $(i, j) \leftarrow (k, t)$

**end-for**

$\text{LSS}(A[1..n]) = A[i..j]$

**return**  $A[i..j]$

Post-Cond: output is longest  $\Delta$ -smooth subarray of  $A[1..n]$ .

**end**

## Algorithm LongestSmoothSubArray(A[1..n], Δ)      § takes O(n) time

```
(i, j, k)  $\leftarrow$  (0, 0, 0); (MinL, MaxL)  $\leftarrow$  ( $\emptyset$ ,  $\emptyset$ )
for t  $\leftarrow$  1 .. n do
```

$$MP = 3(n - t) + |MinL| + |MaxL|$$

```
while MinL  $\neq$   $\emptyset$  & A[t] - Δ > A[Rear(MinL)] do k  $\leftarrow$  1 + PopRear(MinL)
while MaxL  $\neq$   $\emptyset$  & A[t] + Δ > A[Rear(MaxL)] do PopRear(MaxL)
```

```
while MaxL  $\neq$   $\emptyset$ 
while MinL  $\neq$   $\emptyset$  https://eduassistpro.github.io/ k  $\leftarrow$  1 + PopRear(MaxL)
                                         ^ k  $\leftarrow$  1 + PopRear(MinL)
```

```
while MinL  $\neq$   $\emptyset$  & A[t]  $\leq$  A[Front(MinL)]
while MaxL  $\neq$   $\emptyset$  & A[t]  $\geq$  A[Front(MaxL)]
PushFront(t, MinL); PushFront(t, MaxL)
```

```
if j - i < t - k then (i, j)  $\leftarrow$  (k, t)
end-for

return A[i..j]
```

```
end
```

# RECURSION

Assignment Project Exam Help

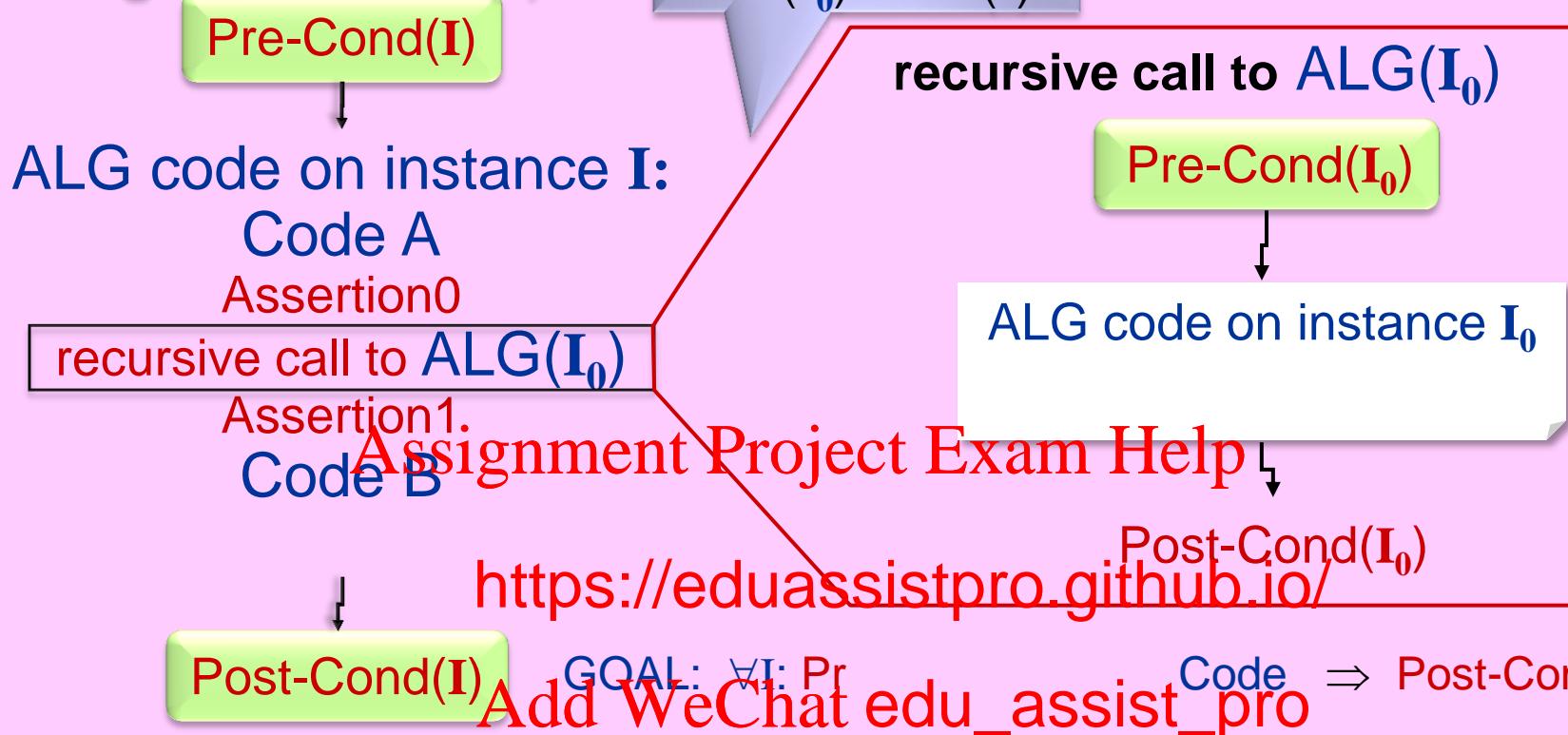
<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

# TOPICS

- Recursive Pre- & Post- Conditions
- Correctness through Strong Induction
- Divide-&-Conquer:
  - Recurs
    - Exp <https://eduassistpro.github.io/>
    - Fibonacci Numbers [Add WeChat edu\\_assist\\_pro](#)
  - Euclid's GCD algorithm
  - Integer & Matrix Multiplication
  - Closest Pair of Points in the plane
  - Algorithms on Binary Trees

# Algorithm ALG(I)

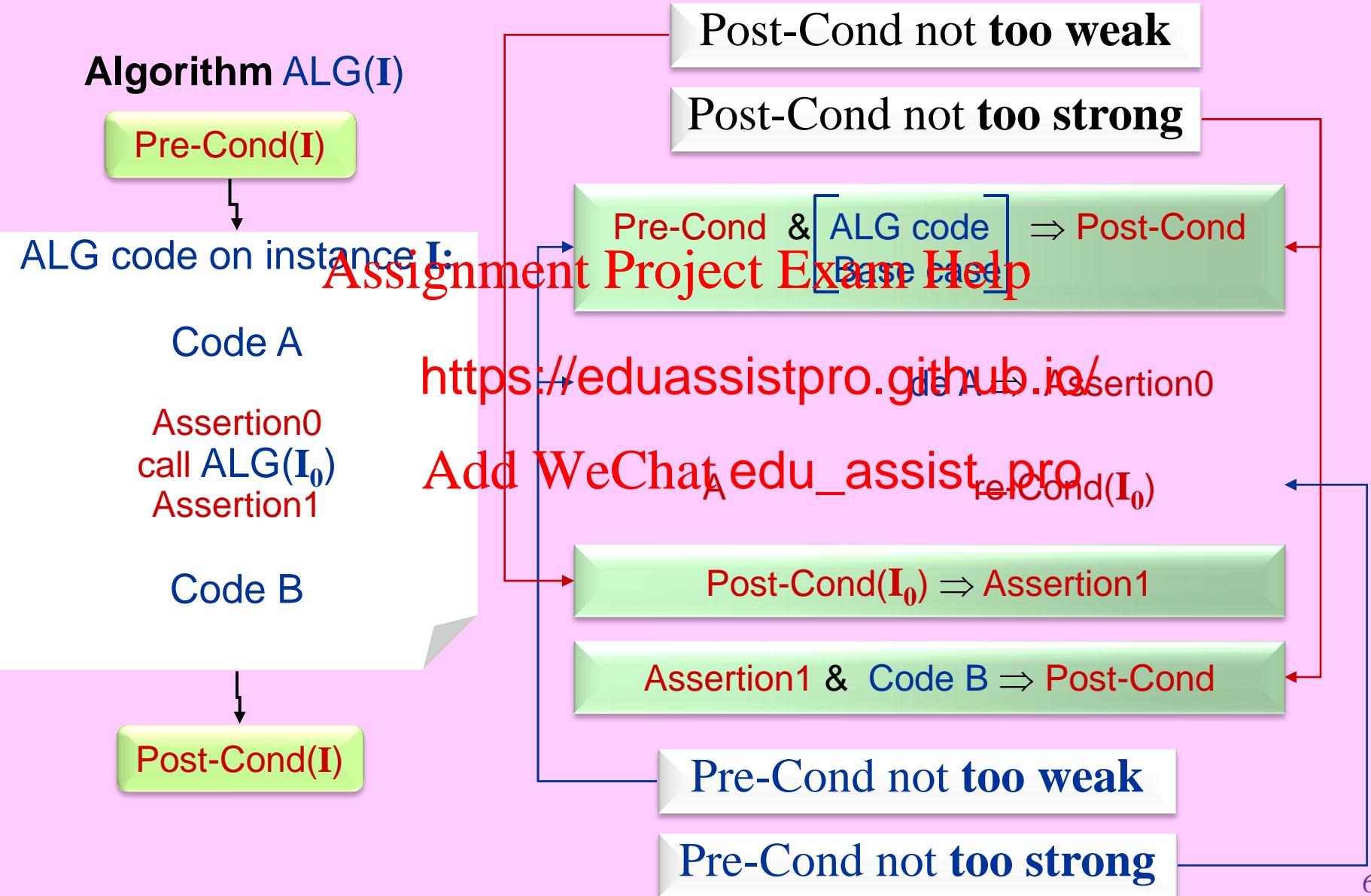


Assume strong induction hypothesis: Pre-Cond(I<sub>0</sub>) & ALG code(I<sub>0</sub>)  $\Rightarrow$  Post-Cond(I<sub>0</sub>)

Prove for an arbitrary instance I:

- Pre-Cond & ALG code Base case  $\Rightarrow$  Post-Cond  $\xleftarrow{\text{induction base}}$
- Pre-Cond & Code A  $\Rightarrow$  Assertion0
- Assertion0  $\Rightarrow$  Pre-Cond(I<sub>0</sub>)
- Post-Cond(I<sub>0</sub>)  $\Rightarrow$  Assertion1
- Assertion1 & Code B  $\Rightarrow$  Post-Cond

# Pre-Cond, Post- Cond, or code may need revision



# DIVIDE

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

**Divide et impera.**  
- Niccolò Machiavelli [1469-1527]

# The Divide-&-Conquer Method

**MergeSort** is the prototypical divide-&-conquer algorithm.

**Algorithm ALG(I)**

Pre-Condition: input is instance I

Post-Condition: ~~ALG(I) is Sol(I)~~ Assignment Project Exam Help

Base:      **if**  $|I| = O(1)$       e method

Divide:     Divide I in <https://eduassistpro.github.io/>      ch of size  $|I|/b$

Conquer:    **for**  $i \leftarrow 1 \dots a$  **do**  $Sol_i \leftarrow ALG(Sol_i)$

Combine:    **Add WeChat** [edu\\_assist\\_pro](https://edu_assist_pro)  
 $Sol \leftarrow \text{Combine}(Sol_1, Sol_2, \dots)$

Output:     **return** Sol

**end**

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + f(n) & \forall n > \Omega(1) \\ \Theta(1) & \forall n \leq O(1) \end{cases}$$

# What Number did I guess?

## Problem 1:

I guess an integer  $x$  between 1 and  $n$ . You find  $x$ .

You can ask me questions of the form “is  $x=a?$ ”, “is  $x > a?$ ”, “is  $x < a?$ ”, for any number  $a$  you choose.

What's your method? How many questions do you need to ask?

Do binary search on integers between 1 and  $n$ .  $O(\log n)$  questions.

<https://eduassistpro.github.io/>

## Problem 2:

Add WeChat `edu_assist_pro`

I guess a positive integer  $x$ . You find  $x$

You can ask me questions of the form “is  $x=a?$ ”, “is  $x > a?$ ”, “is  $x < a?$ ”, for any number  $a$  you choose.

What's your method? How many questions do you need to ask?

Starting from 1, repeatedly double to get an upper-bound  $n$  on  $x$ .

Then binary search between 1 and  $n$ .  $O(\log x)$  questions.

# Recursive Doubling

**Example:** Given  $X \in \mathbb{R} - \{0\}$  and  $n \in \mathbb{N}$ , compute  $X^n$  using arithmetic.

**Method 1:** Multiply  $n$  copies of  $X$ :  $X \cdot X \cdot X \cdots X \cdot X \cdot X$ .

That takes  $O(n)$  arithmetic operations.

That's **exponential** in the bit length of input  $n$ .

**Method 2:** [Assignment Project Exam Help](https://eduassistpro.github.io/)

Starting with  $X$ , repeatedly multiply the result by itself. We get:

$$X, X^2, X^4, \dots$$

Suppose we want to compute  $X^{13}$ . So,  $X^{13} =$

$X^n$  can be computed this way in  $O(\log n)$  arithmetic operations. How?

Just figure out  $n$  in binary bits.

It's simpler to **look backwards**. Divide  $n$  by 2 and let  $r$  be the remainder:

$$n = 2 \cdot \left\lfloor \frac{n}{2} \right\rfloor + r, \quad r \in \{0,1\}.$$

$$x^n = x^{2 \lfloor \frac{n}{2} \rfloor + r} = x^{\frac{n}{2}} \cdot x^r = y^m \cdot x^r$$

$$\text{where } y = x \cdot x \text{ and } m = \left\lfloor \frac{n}{2} \right\rfloor.$$

# $X^n$ by Recursive Doubling

**Algorithm EXP(X, n)**

Pre-Cond:  $X \in \mathbb{R} - \{0\}$ ,  $n \in \mathbb{N}$ .

**if**  $n=0$  **then return** 1

$Y \leftarrow X \cdot X$

$Z \leftarrow \text{EXP}(Y, \lfloor n/2 \rfloor)$

**if**  $n \bmod 2 = 1$  **then**  $Z \leftarrow Z \cdot X$

**return** Z

$$n = 2 \cdot \left\lfloor \frac{n}{2} \right\rfloor + r, \quad r \in \{0,1\}.$$

$$x^n = x^{2\lfloor \frac{n}{2} \rfloor + r} = y^{\lfloor \frac{n}{2} \rfloor} \cdot x^r$$

Post-Cond: returns

**end**

$$r = n \bmod 2.$$

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

$$T(n) = \begin{cases} T(\frac{n}{2}) + \Theta(1) & \forall n > 0 \\ \Theta(1) & n = 0 \end{cases} \Rightarrow T(n) = \Theta(\log n).$$

# Fibonacci Numbers

## Fibonacci Numbers:

$$F_0 = 0, \quad F_1 = 1, \quad F_{n+2} = F_{n+1} + F_n \quad \text{for all } n \geq 0.$$

n:	0	1	2	3	4	5	6	7	8	9	10	11	12	...
$F_n$ :	0	1	1	2	3	5	8	13	21	34	55	89	144	...

Assignment Project Exam Help

<https://eduassistpro.github.io/>

$$x^{n+2} = x^{n+1} + x^n, \quad x \neq 0 \Leftrightarrow x^2 = x + 1.$$

Two roots :

Add WeChat edu\_assist\_pro

$$\text{golden ratio } \phi = \frac{1+\sqrt{5}}{2} = 1.61803\cdots,$$



$$F_n = \frac{1}{\sqrt{5}} (\phi^n - \hat{\phi}^n)$$

$$\text{and } \hat{\phi} = \frac{1-\sqrt{5}}{2} = -0.61803\cdots.$$

$$F_n = a \cdot \phi^n + b \cdot \hat{\phi}^n$$

Now choose  $a$  and  $b$  to satisfy the

boundary conditions  $F_0 = 0, F_1 = 1$ .



$$F_n = \Theta(\phi^n)$$

# Method 1: the recurrence formula

**Algorithm Fib(n)**

```
if n∈{0,1} then return n  
return Fib(n-1) + Fib(n-2)  
end
```

Assignment Project Exam Help

$$T(n) = \begin{cases} T(n-1) + T(n-2) + \Theta(1) & n \geq 2 \\ \Theta(1) & n \in \{0,1\} \end{cases}$$

⇒  $T(n) = \Theta(F_n)$  [WeChat edu\_assist\_pro  
guess & verify]

Cons: This is double-exponential time  
in the input n's bit length!

## Method 2: incremental

**Algorithm Fib(n)**

$(x,y) \leftarrow (0,1) \quad \S \quad (x,y) = (F_0, F_1)$

**for**  $t \leftarrow 2 .. n$  **do**

$z \leftarrow x + y$

$x \leftarrow y$

$y \leftarrow z$

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

$T(n) = \Theta(n).$

Cons: This is exponential time  
in the input n's bit length!

# Method 3: recursive doubling

$$F_n = \frac{1}{\sqrt{5}} (\phi^n - \hat{\phi}^n)$$

**Algorithm Fib(n)**

X ← EXP( (1 + √5)/2 , n)

Y ← EXP( (1 - √5)/2 , n)

return (X - Y)/√5

**end**

Assignment Project Exam Help

$T(n) = \Theta(\log n)$ .

Pros: This algorithm has a time complexity of  $\Theta(\log n)$ , where  $n$  is the bit length!

Add WeChat edu\_assist\_pro

Cons:

- What if we didn't know the formula?
- It uses non-integer real arithmetic &  $\sqrt{\cdot}$  even though  $F_n$  is always an integer!

Is there a way to do integer-only arithmetic to avoid “round off” errors?

## Method 4: $A^n$ recursive doubling

$$\begin{aligned}F_{n+1} &= F_n + F_{n-1} \\F_n &= F_n\end{aligned}$$



$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix}$$

$$\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix}$$

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}}_{G(n)} \underbrace{\begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{pmatrix}}_{G(1)} \underbrace{\begin{pmatrix} F_1 & F_0 \\ F_1 & F_0 \end{pmatrix}}_{G(0)} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = A$$

Assignment Project Exam Help  
<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

$$\rightarrow G(n) = A \cdot G(n-1) = A^2 \cdot G(n-2) \quad (1) = A^n .$$

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \quad \forall n \geq 1$$

**EXP(A , n)**      on 2-by-2 matrices!  
O(log n) time integer-only arithmetic!

# Greatest Common Divisor

**Input:**  $a, b \in \mathcal{N}$ , not both zero.

**Output:** Greatest Common Divisor of  $a$  and  $b$ :  
 $\text{GCD}(a,b) = \max \{ d \in \mathcal{N} : d|a \text{ and } d|b \}.$

$d|a$  means “ $d$  divides  $a$ ”, i.e.,  $a$  is an integer multiple of  $d$ .

Assignment Project Exam Help

Elementary

if  $mi \leftarrow \min\{a,b\}$

for  $d \leftarrow mi$  do

if  $d|a \& d|b$  then return  $d$

end

Running time =  $O(1 + \min\{a,b\})$ .

This is **exponential** time in the # bits in the input.

Try  $\text{GCD}(1000,000,000,001, 1000,000,000,000)$ .

# Euclid's GCD Algorithm

**FACT 0:**  $d|a \& d|b \Leftrightarrow d|\text{GCD}(a,b)$ .

**FACT 1:**  $b = 0 \Rightarrow \text{GCD}(a,b) = a$ .

Now suppose  $b \neq 0$ . Divide a by b.

Let  $q$  = quotient,  $r$  = remainder:

$$a = q \cdot b + r, \quad 0 \leq r < b$$

**FACT 2:**  $\text{GCD}(a,b) = \text{https://eduassistpro.github.io/Assignment Project Exam Help}$

Proof:  $\forall d: d|a \& d|b \Leftrightarrow d|b \& d|r$ .

**FACT 3:**

(a)  $a < b \Rightarrow q = 0, r = a$ . So,  $(b,r) = (b,a)$ .

(b)  $a = b \Rightarrow q = 1, r = 0$ . So,  $(b,r) = (b,0)$ .

(c)  $a > b \Rightarrow a \geq b + r > 2r$ .

$$r_0 = a, \quad r_1 = b,$$

For all  $i$ , such that  $r_{i+1} \neq 0$ :

$$r_i = q_i r_{i+1} + r_{i+2}, \quad 0 \leq r_{i+2} \leq r_{i+1} - 1.$$

**Algorithm  $\text{GCD}(a,b)$**

```
if b=0 then return a  
return  $\text{GCD}(b, a \bmod b)$   
end
```

$$(r_0, r_1) = (a, b)$$

$$(r_2, r_3) \\ (r_4)$$

$$\dots \\ (r_k, r_{k+1}), \quad r_{k+1} = 0 \\ r_k = \text{GCD}(a,b)$$

**FACT 4:**

- (a) For all  $i > 0$ :  $r_{i+2} < \frac{1}{2} r_i$ .
- (b)  $k < 2 + 2 \log \min\{a,b\}$ .
- (c) Running time of GCD  
 $= O(1 + \log \min\{a,b\})$ .

# More on GCD

**FACT 5:** The following are equivalent:

$$(1) D = \text{GCD}(a,b)$$

$$(2) D = \max \{d \in \mathbb{N} : d|a \text{ and } d|b\}.$$

$$(3) D = \min \{ax + by > 0 : x \text{ & } y \text{ are integers}\}.$$

**Proof:**  $(1) \Leftrightarrow (2)$  by definition. We show  $(1) \Leftrightarrow (3)$  as follows:

Let  $D = \text{GCD}(a,b)$  a & y are integers}.

$D|a \text{ & } D|b \Rightarrow D|(ax + by)$  <https://eduassistpro.github.io/>

$D \geq D'$  is implied by the Claim below.  $D'$ .

**CLAIM:**  $D = \text{GCD}(a,b) = ax + by$ , for some integers x and y.

**Proof of Claim:** By induction on the recursion depth of Euclid's algorithm.

**Basis:** If  $b=0$ , then  $\text{GCD}(a,b) = a = 1a + 0b$ .

**Induction:** If  $b \neq 0$ , let  $a = qb + r$ .

By induction hypothesis  $\text{GCD}(b, r) = bx' + ry'$ , for some integers  $x'$  &  $y'$ .  
Therefore,  $\text{GCD}(a,b) = \text{GCD}(b, r) = bx' + ry' = bx' + (a - qb)y' = ay' + (x' - qy')b = ax + by$   
with  $x = y'$  and  $y = x' - qy'$ .

# More on GCD

**FACT 5:** The following are equivalent:

- (1)  $D = \text{GCD}(a,b)$
- (2)  $D = \max \{d \in \mathbb{N} : d|a \text{ and } d|b\}$ .
- (3)  $D = \min \{ax + by > 0 : x \text{ & } y \text{ are integers}\}$ .

**FACT 6:** Assignment Project Exam Help  
For (possibly negative) integers  $a$  and  $b$ ,

$$\text{GCD}(a,b) = \text{GCD}(|a|, |b|)$$

<https://eduassistpro.github.io/>

**FACT 7:** The equa

with given non-zero integers  $a$  and  $b$ , has integer solutions for  $x$  and  $y$   
if and only if  $a$  and  $b$  are relatively prim

$$\text{GCD}(a,b) = 1.$$

**FACT 8:** The equation  $ax + by = c$

with given non-zero integers  $a, b, c$ , has integer solutions for  $x$  and  $y$   
if and only if  $\text{GCD}(a,b)|c$ .

**Exercise:**

Show that Euclid's algorithm can be adapted to produce such a solution.

# n-bit Integer Addition vs Multiplication

$$x = x_{n-1}x_{n-2} \cdots x_1x_0$$

compute  $x + y$

$$y = y_{n-1}y_{n-2} \cdots y_1y_0$$

and  $x * y$

**FACT 0:** The bit-complexity of n-bit addition or multiplication is  $\Omega(n)$ .

**Proof:** A correct algorithm must “look” at every input bit.

Suppose on non-zero inputs, input bit b is not looked at by the algorithm.

Adversary gives the algorithm the same input, but with bit b flipped.

Algorithm is oblivious to the value of b, so it will give the same answer.

It can't be correct bot <https://eduassistpro.github.io/>

Elementary School Add We All At [edu\\_assist\\_pro.com](https://edu_assist_pro.com) complexity:

$$\begin{array}{r} \text{XXXXXXXXXX} \\ + \text{XXXXXXXXXX} \\ \hline \text{XXXXXXXXXX} \end{array}$$

**FACT 1:** The bit-complexity of n-bit addition is  $\Theta(n)$ .

# n-bit Integer Multiplication

$$x = x_{n-1}x_{n-2} \cdots x_1x_0$$

compute  $x + y$

$$y = y_{n-1}y_{n-2} \cdots y_1y_0$$

and  $x * y$

Elementary School Multiplication Algorithm has  $O(n^2)$  bit-complexity:

XXXX  
\* XXXX  
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro  
XXXXXXXXXX

**QUESTION:** What is the bit-complexity of n-bit multiplication?

# n-bit Integer Multiplication by Divide-&-Conquer

$$x = x_{n-1}x_{n-2} \cdots x_1x_0$$

compute  $x + y$

$$y = y_{n-1}y_{n-2} \cdots y_1y_0$$

and  $x * y$

X = 3141
Y = 5927

$$X * Y = 18,616,707$$

Assignment Project Exam Help

$$X = 3100 + 4$$

<https://eduassistpro.github.io/>

$$Y = 5900 + 2$$

$$X * Y = (a * c) \cdot 10000 + (a * d + b * c) \cdot 100 + b * d$$

$$= (31 * 59) \cdot 10000 + (31 * 27 + 41 * 59) \cdot 100 + 41 * 27$$

$$= 1829 \cdot 10000 + (837 + 2419) \cdot 100 + 1107$$

$$= 1829 \cdot 10000 + 3256 \cdot 100 + 1107$$

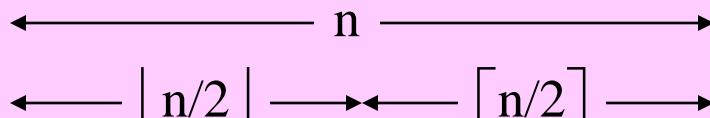
$$= 18290000 + 325600 + 1107$$

$$= 18,616,707$$

# n-bit Integer Multiplication by Divide-&-Conquer

$$x = x_{n-1}x_{n-2} \cdots x_1x_0$$
$$y = y_{n-1}y_{n-2} \cdots y_1y_0$$

compute  $x + y$   
and  $x * y$



$x =$  Assignment Project Exam Help

$y =$  https://eduassistpro.github.io/

$$x * y = (a * c) \cdot 2^{2\lceil \frac{n}{2} \rceil} + (a * d + c * b) \cdot 2^{\lceil \frac{n}{2} \rceil} + (b * d)$$

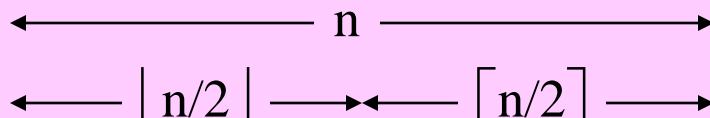
Shift/Add/Subtract  $\Theta(n)$  time

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

# n-bit Integer Multiplication by Divide-&-Conquer

$$x = x_{n-1}x_{n-2} \cdots x_1x_0$$
$$y = y_{n-1}y_{n-2} \cdots y_1y_0$$

compute  $x + y$   
and  $x * y$



$x =$  Assignment Project Exam Help

$y =$  https://eduassistpro.github.io/

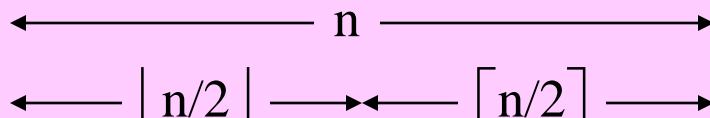
$$x * y = (a * c) \cdot 2^{2\lceil \frac{n}{2} \rceil} + (a * d + c * b) \cdot 2^{\lceil \frac{n}{2} \rceil} + (b * d)$$

Shift/Add/Subtract  $\Theta(n)$  time

$$(a_n + b) * (c_{\frac{n}{2}} + d) = (a * c) + (a * d + c * b) + (b * d)$$

# n-bit Integer Multiplication by Divide-&-Conquer

$$x = x_{n-1}x_{n-2} \cdots x_1x_0 \quad \text{compute } x + y$$
$$y = y_{n-1}y_{n-2} \cdots y_1y_0 \quad \text{and } x * y$$



$x =$  Assignment Project Exam Help

$y =$  https://eduassistpro.github.io/

Divide-&-Conquer Multiplication, Ofman [1962]:  
Add WeChat edu\_assist\_pro

$u \leftarrow (a + b) * (c + d)$  Karatsu Ofman's theory disproved  
 $v \leftarrow a * c$  Andrey Kolmogorov's 1956 conjecture that  
 $w \leftarrow b * d$  multiplication requires  $\Omega(n^2)$  bit operations.  
 $xy \leftarrow v \cdot 2^{[n/2]} + (u - v - w) \cdot 2^{[n/2]} + w$   
return  $xy$

$$T(n) = 3T(\frac{n}{2}) + \Theta(n) \Rightarrow T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.585\dots})$$

# n-bit Integer Multiplication by Divide-&-Conquer

$$x = x_{n-1}x_{n-2} \cdots x_1x_0$$
$$y = y_{n-1}y_{n-2} \cdots y_1y_0$$

compute     $x + y$   
and             $x * y$

$$X = 3141$$
$$Y = 5927$$

$$X * Y = 18,616,707$$

Assignment Project Exam Help

$$X = 3100 + 4$$
$$Y = 5900 + 2$$

$$u = (a+b)*(c+d) = (31+41)*(59+2) = 6,192$$

$$v = a*c = 31 * 59 = 1,829$$

$$w = b*d = 41 * 27 = 1,107$$

$$X * Y = v * 10000 + (u - v - w) * 100 + w$$

$$= 1829 * 10000 + (6192 - 1829 - 1107) * 100 + 1107$$

$$= 18,616,707$$

# Complexity of $n$ -bit Integer Multiplication

## Known Upper Bounds:

- $O(n^{\log 3}) = O(n^{1.59})$  by divide-&-conquer [Karatsuba-Ofman, 1962]
- $O(n \log n \log \log n)$  by FFT [Schönhage-Strassen, 1971]
- $O(n \log n 8^{\log^* n})$  vid Harvey et al., 2016]

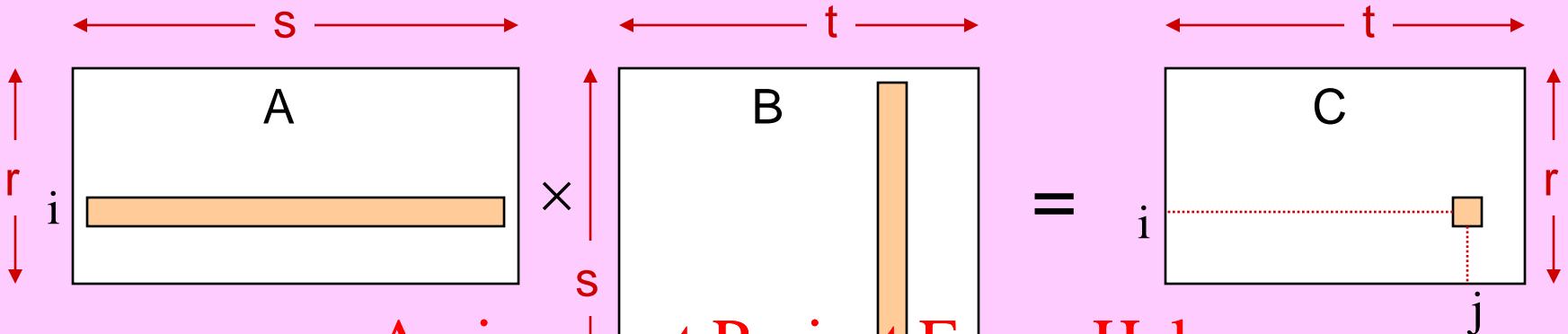
<https://eduassistpro.github.io/>

## Known Lower Bound: Add WeChat edu\_assist\_pro

- $\Omega(n \log n / \log \log n)$  [Fischer-Meyer, 1974]

FFT = Fast Fourier Transform [CLRS chapter 30]

# Matrix Multiplication



Assignment Project Exam Help

<https://eduassistpro.github.io/>

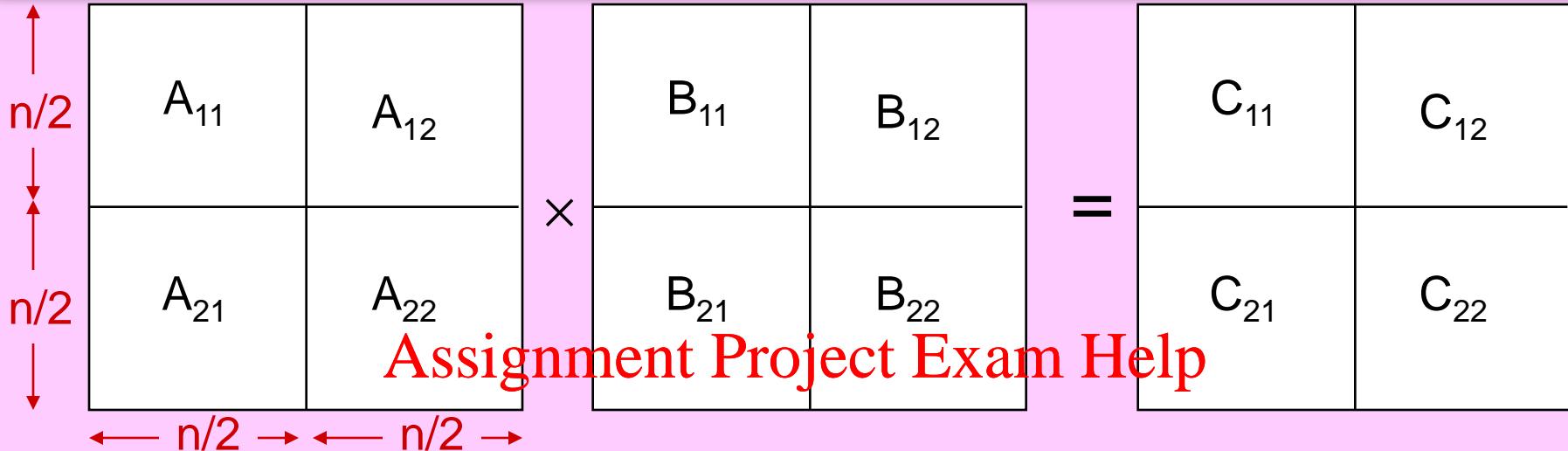
Add WeChat edu\_assist\_pro

$$C_{ij} = \sum_{k=1}^s A_{ik} \cdot B_{kj} \quad \text{for } i = 1..r, j = 1..t.$$

$\Theta(rst)$  time.

$\Theta(n^3)$  time, if  $r = s = t = n$ .

# Matrix Multiplication by Divide-&-Conquer



<https://eduassistpro.github.io/>

$$C_{11} = A_{11}B_{11} + A_{12}B_{21} \quad C_{12} = A_{11}B_{12} + A_{12}B_{22}$$
$$C_{21} = A_{21}B_{11} + A_{22}B_{21} \quad C_{22} = A_{21}B_{12} + A_{22}B_{22}$$

$$T(n) = 8T(\frac{n}{2}) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^{\log_2 8}) = \Theta(n^3)$$

# Strassen's Matrix Multiplication Algorithm

$$\begin{array}{|c|c|} \hline A_{11} & A_{12} \\ \hline A_{21} & A_{22} \\ \hline \end{array} \times \begin{array}{|c|c|} \hline B_{11} & B_{12} \\ \hline B_{21} & B_{22} \\ \hline \end{array} = \begin{array}{|c|c|} \hline C_{11} & C_{12} \\ \hline C_{21} & C_{22} \\ \hline \end{array}$$

$m_1 = (A_{11} + A_{22})(B_{11} + B_{22})$  Assignment Project Exam Help

$m_2 = (A_{11} + A_{22})(B_{11} + B_{22})$   $C_{12} = m_4 +$

$m_3 = (A_{11} - A_{21})(B_{11} + B_{12})$   $C_{21} = m_6 +$

$m_4 = (A_{11} + A_{12})B_{22}$   $C_{12} = m_2 - m_3 + m_5 - m_7$  Add WeChat **edu\_assist\_pro**

$m_5 = A_{11}(B_{12} - B_{22})$

$m_6 = A_{22}(B_{21} - B_{11})$

$m_7 = (A_{21} + A_{22})B_{11}$

Strassen's discovery broke the  $\Omega(n^3)$  barrier.

There has been further recent improvements.

$$T(n) = 7T(\frac{n}{2}) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81...})$$

# The Closest Pair of Points

- Input:** A set  $P = \{p_1, p_2, \dots, p_n\}$  of  $n$  points in the plane.  
Each point is given by its  $x$  &  $y$  coordinates  $p_i = (x_i, y_i)$ ,  $i=1..n$ .
- Output:** The closest pair of points in  $P$ , i.e., the pair  $(p_i, p_j)$  in  $P$ ,  $i \neq j$ ,  
with minimum Euclidean distance  $d(p_i, p_j) = ((x_i - x_j)^2 + (y_i - y_j)^2)^{1/2}$ .



**In 1 dimension: The Min-Gap Problem.**

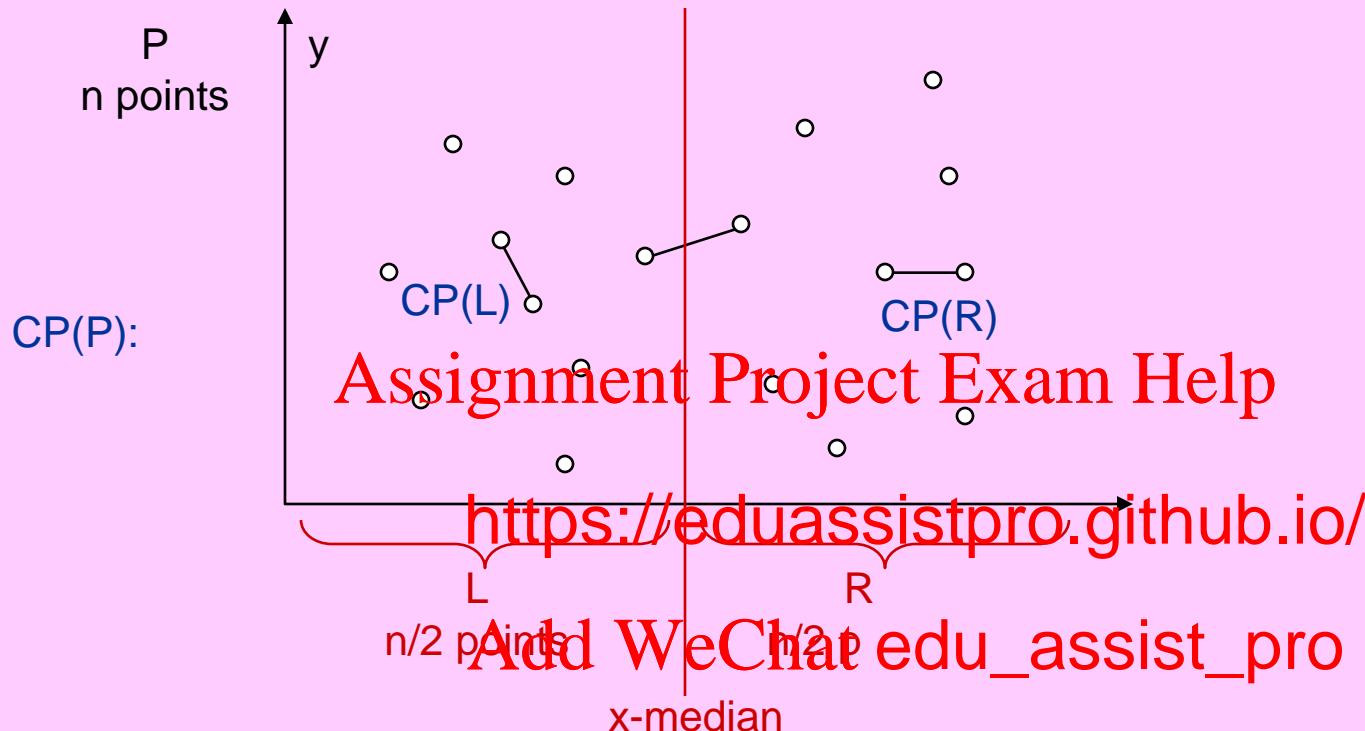
Known lower bound on time complexity:  $\Omega(n \log n)$ . Discussed later.  
Matching upper bound (by sorting first):  $O(n \log n)$ .

**In 2 dimensions** it is at least as hard.

**Brute Force:** Try every pair. That takes  $\Theta(n^2)$  time.

**Divide-&-Conquer** ..... P.T.O.

# CP: Divide-&-Conquer



**3 possible cases for the closest pair of points:**

- a) both in the left half L, [recursive call on L]
- b) both in the right half R, [recursive call on R]
- c) one in L and the other in R. [Combine L & R]

# Divide-&-Conquer template

## Algorithm **ClosestPair(P)**

Pre-Condition: P is a finite set of points in the plane

Post-Condition: Output is the closest pair of points in P

1. Pre-Sort points in P on their x-coordinates (lexicographically next on y)
2. **return CP(P)**

**end**

[Assignment](#) [Project](#) [Exam](#) [Help](#)

## Procedure **CP(P)** <https://eduassistpro.github.io/>

Pre-Condition: P is a x-sorted finite set o

Post-Condition: Output is the closest pair

3. **Base:** **if**  $|P| < 10$  **then return** answer by brute-force in  $\Theta(1)$  time
  4. **Divide:** Partition P at its x-median value into sets L and R,  $|L| \approx |R| \approx |P|/2$
  5. **Conquer:**  $SolL \leftarrow CP(L); SolR \leftarrow CP(R)$
  6. **Combine:**  $Sol \leftarrow MERGE(SolL, SolR)$
  7. **Output:** **return** Sol
- end**

[Add WeChat edu\\_assist\\_pro](#)

# Divide-&-Conquer template

**Algorithm** Post-Cond of MERGE must imply Post-Cond of CP(P).

Pre-Condition: P is a finite set of points in the plane  
On the other hand,

Post-Condition: Output is the closest pair of points in P

1. Pre-Sort points in P by their x-coordinates (lexicographically next on y)

2. return CP(P)

end Strengthen Post-Cond of CP (CP(L) & CP(R))  
**Assignment Project Exam Help**  
to help reduce the burden on MERGE!

**Procedure** CP(P) <https://eduassistpro.github.io/>

Pre-Condition: P is a x-sorted finite set o

Post-Condition: Output is the closest pair

3. Base: if  $|P| < 10$  then return answer by brute-force in  $\Theta(1)$  time

4. Divide: Partition P at its x-median value into sets L and R,  $|L| \approx |R| \approx |P|/2$

5. Conquer:  $SolL \leftarrow CP(L); SolR \leftarrow CP(R)$

6. Combine:  $Sol \leftarrow MERGE(SolL, SolR)$

7. Output: return Sol

end

Our aim

$$T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n) \text{ time.}$$

# Strengthen CP Post-Condition

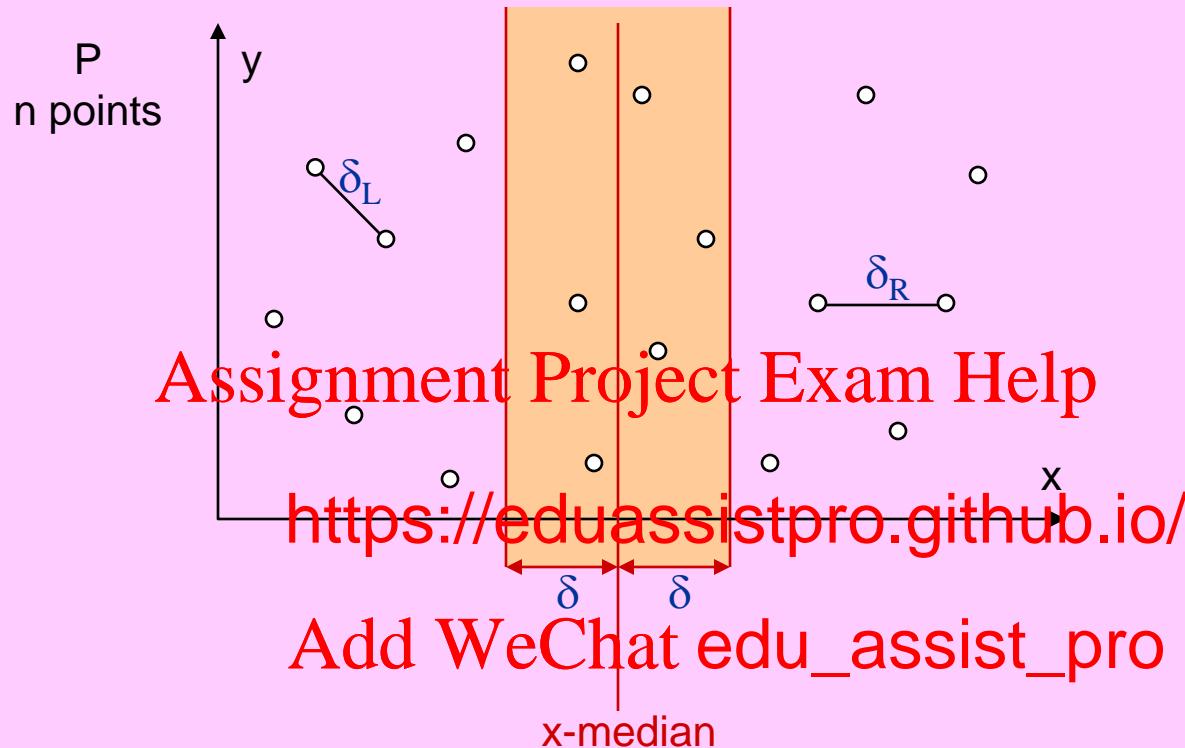
## Procedure CP(P)

Pre-Condition: P is a x-sorted finite set of points

Post-Condition: Output is the closest pair of points in P, and  
P is rearranged into y-sorted order.

- 3.Base:      **if**  $|P| \leq 10$  **then return** answer by brute force in  $\Theta(1)$  time
- 4.Divide:      Partition    L and R,  $|L| \approx |R| \approx |P|/2$
- 5.Conquer:
  - § Now <https://eduassistpro.github.io/>
  - SolL  $\leftarrow$  C
    - § Now A & R are y-sorted
    - § MERGE can y-merge
- 6.Combine:      Sol  $\leftarrow$  MERGE(SolL , SolR )
  - § Now P = L  $\cup$  R is y-sorted, and ...
- 7.Output:      **return** Sol
- end**

# MERGE



MERGE(L, R):

$$\delta = \min \{\delta_L, \delta_R\}$$

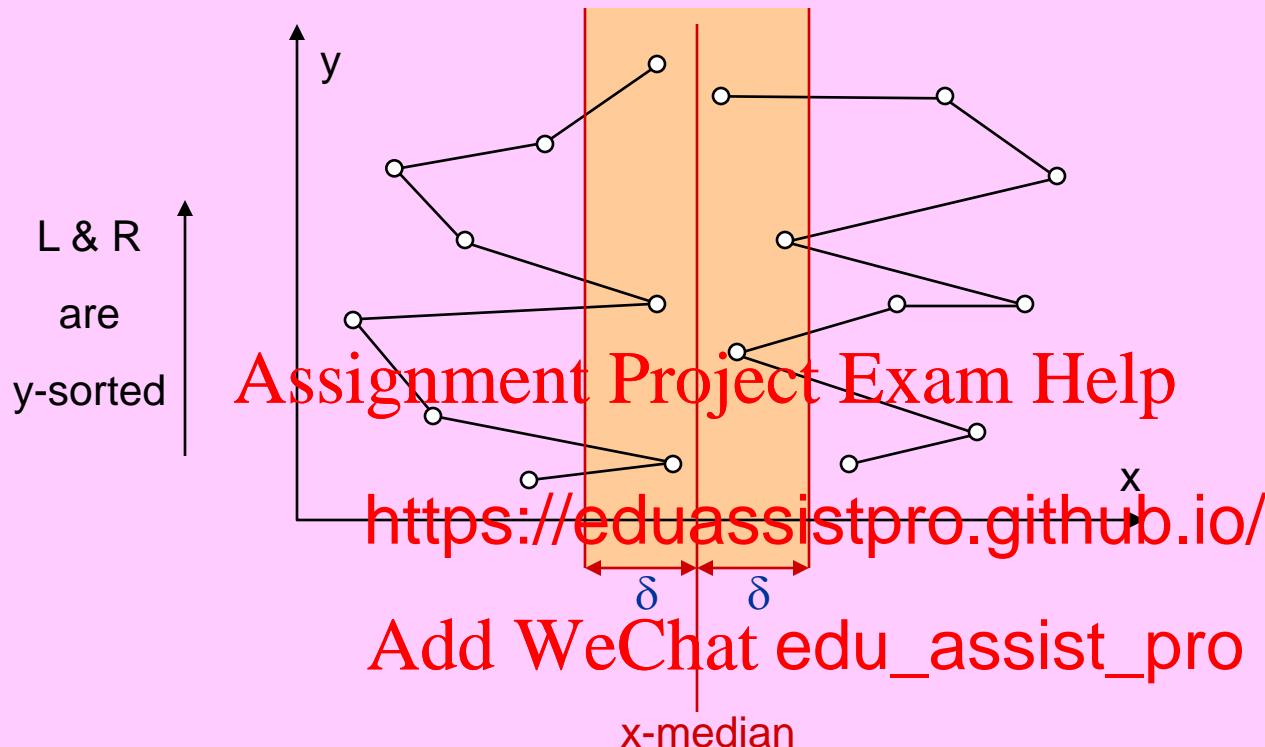
...

...

end

Can we do it  
in O(n) time?

# MERGE



MERGE( $L, R$ ):

$$\delta = \min \{\delta_L, \delta_R\}$$

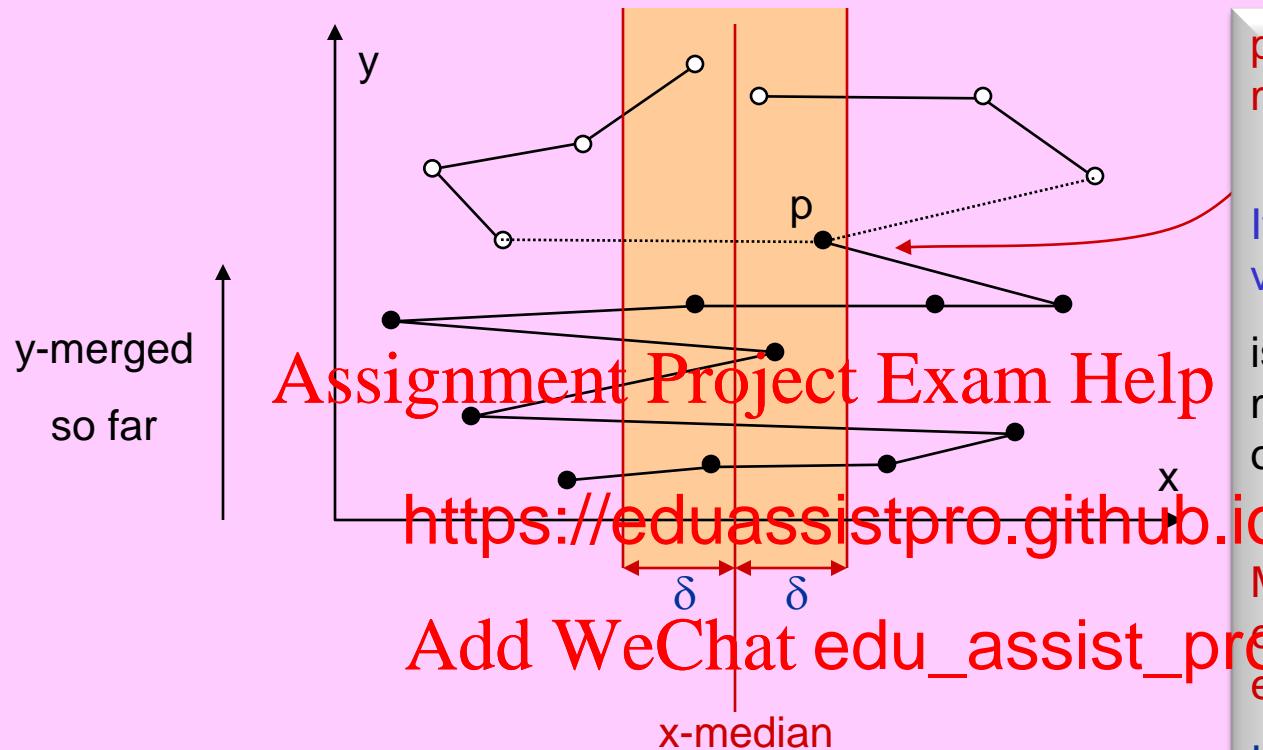
...

...

**end**

Can we do it  
in  $O(n)$  time?

# MERGE



MERGE( $L, R$ ):

$$\delta = \min \{\delta_L, \delta_R\}$$

y-merge  $L$  &  $R$

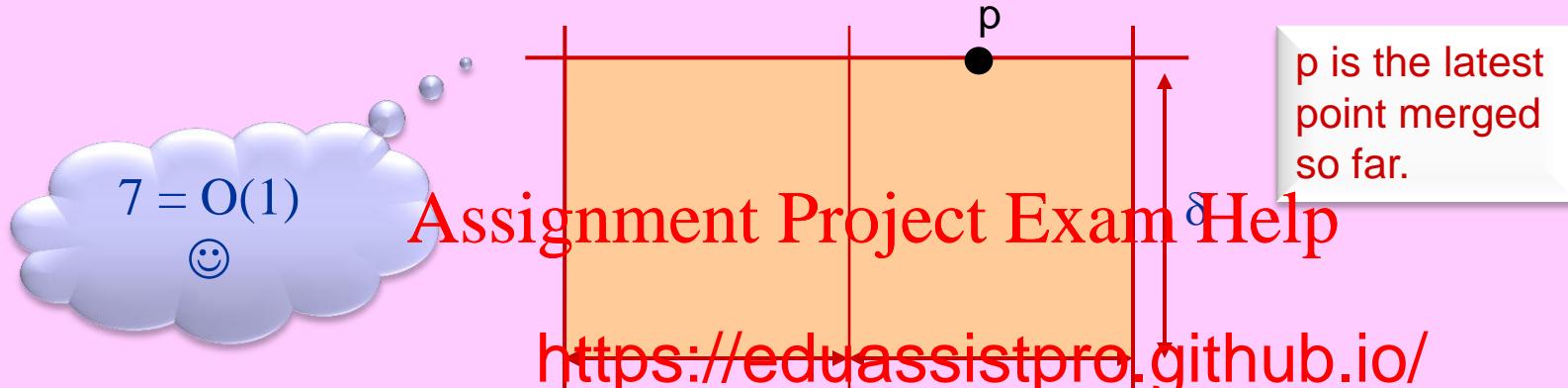
...

**end**

Can we do it  
in  $O(n)$  time?

# MERGE

**FACT:** There can be at most 7 points (excluding p) in the shaded  $2\delta$ -by- $\delta$  rectangle shown below. Why?



**MERGE:**

- Maintain the (up to) 7 latest merged points that fall within the  $2\delta$  vertical slab.
- If next point  $p$  being merged falls within this slab  
then compare  $p$  against the “7 points”; update closest pair;  
add  $p$  to the “7 point” list (remove the now lowest 8<sup>th</sup> from the list).
- Add  $p$  to the merged list and move up to the next point.
- Each point is y-merged in  $O(1)$  time.  
MERGE takes  $O(n)$  time. Therefore, CP takes  $O(n \log n)$  time.

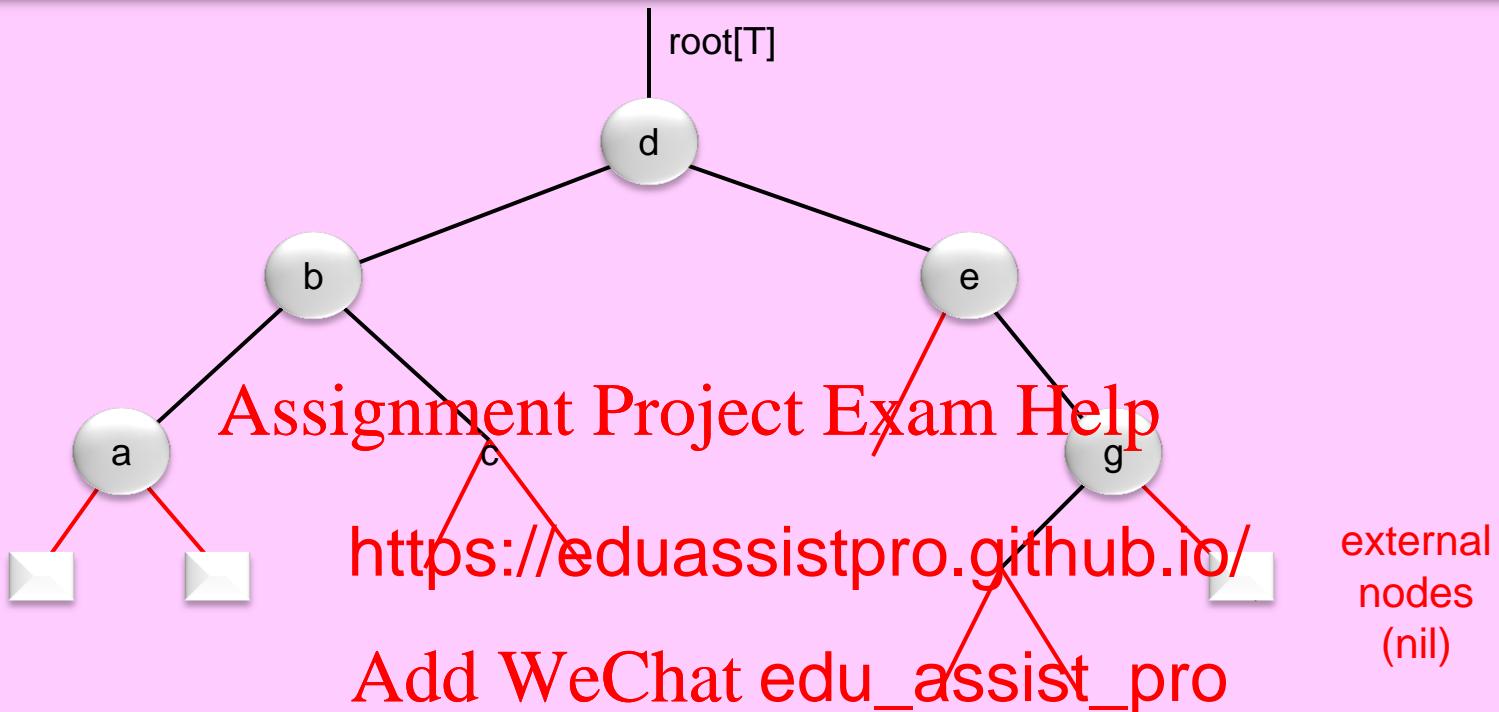
# Binary Trees

Assignment Project Exam Help

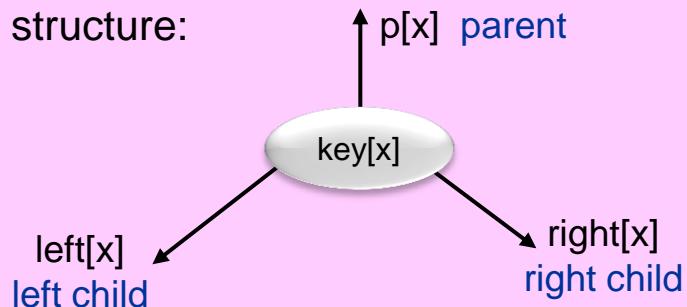
<https://eduassistpro.github.io/>

Binary Sea Add WeChat edu\_assist\_pro rees

# Binary Trees: A Quick Review



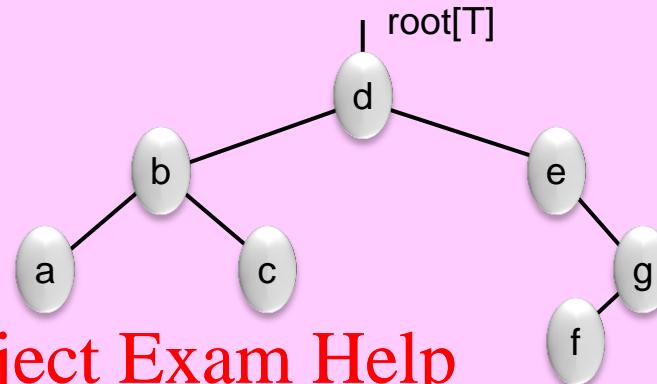
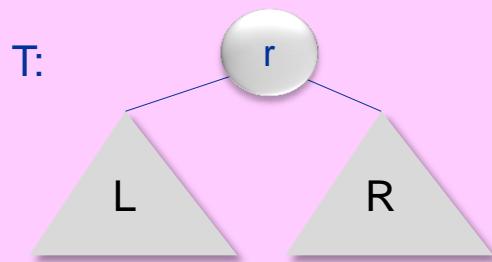
Node  $x$  structure:



$n$  (internal) nodes  
 $n-1$  (internal) edges

$n+1$  external nodes (nil)  
 $n+1$  external edges (nil)

# Binary Tree Traversals

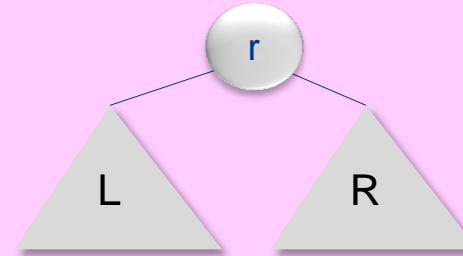


Assignment Project Exam Help

- Inorder( $T$ ):      Inorder <https://eduassistpro.github.io/>
- Preorder( $T$ ):       $r ; \text{Preorder}(L) ; \text{Preorder}(R)$ .  
                        Add WeChat  $\text{edu\_assist\_pro}$  } graph DFS
- Postorder( $T$ ):       $\text{Postorder}(L) ; \text{Postorder}(R) ; r$ .      acbfged }
- Levelorder( $T$ ):      non-decreasing depth order  
(same depth left-to-right)      dbeacgf —————> graph BFS

# Traversals in $O(n)$ time

```
procedure Inorder(x)
1. if x = nil then return
2. Inorder(left[x])
3. visit(x)
4. Inorder(right[x])
end
```



Assignment Project Exam Help

Running Time An <https://eduassistpro.github.io/>

Line 1:  $n+1$  external

s (continue).

Line 3: Assume visit takes  $O(1)$  time

Add WeChat edu\_assist\_pro

Lines 2 & 4: After recursive expansion:

Each node x (internal or external) visited exactly once.

$O(1)$  time execution of lines 1 & 3 charged to node x.

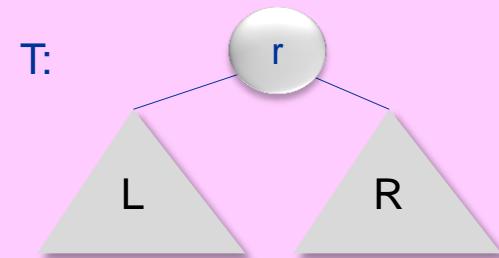
Total  $n + (n+1)$  nodes, each charged  $O(1)$  time.

Total time =  $O(2n+1) = O(n)$ .

- Preorder and Postorder are similar and take  $O(n)$  time.
- **Exercise:** Write a simple  $O(n)$  time algorithm for Levelorder. [Hint: use a queue.]

# Running Time Analysis by Recurrence

$$\text{Time}(T) = \begin{cases} \text{Time}(L) + \text{Time}(R) + 1 & \text{if } T \neq \text{nil} \\ 1 & \text{if } T = \text{nil} \end{cases}$$



**CLAIM:**  $\text{Time}(T) = 2|T| + 1$ . Assignment Project Exam Help

**Proof:** By induction on <https://eduassistpro.github.io/>

**Basis ( $|T|=0$ ):**  $\text{Time}(T) = 1 = 2|T| + 1$ . Add WeChat edu\_assist\_pro

**Induction Step ( $|T| > 0$ ):**

$$\begin{aligned} \text{Time}(T) &= \text{Time}(L) + \text{Time}(R) + 1 && [\text{by the recurrence}] \\ &= (2|L|+1) + (2|R|+1) + 1 && [\text{by the Induction Hypothesis}] \\ &= 2(|L| + |R| + 1) + 1 \\ &= 2|T| + 1. \end{aligned}$$

# BST from Binary Search on Sorted Array

E0 < K1 < E1 < K2 < E2 < K3 < E3 < K4 < E4 < K5 < E5 < K6 < E6 < ... < Kn < En



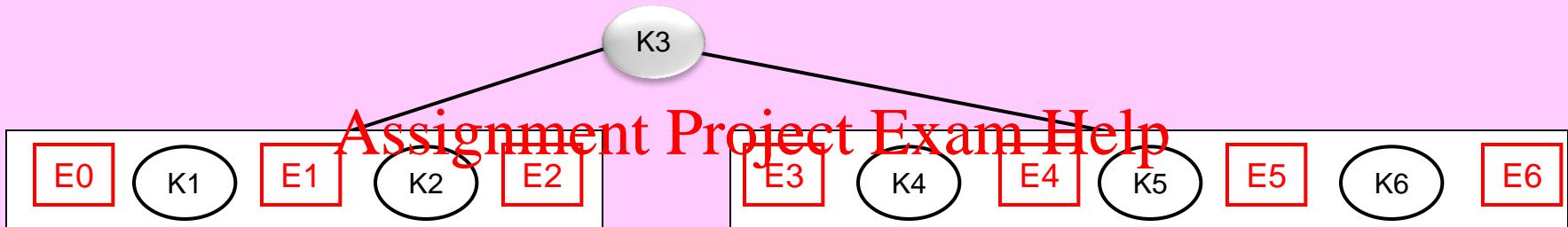
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

# BST from Binary Search on Sorted Array

E0 < K1 < E1 < K2 < E2 < K3 < E3 < K4 < E4 < K5 < E5 < K6 < E6 < ... < Kn < En



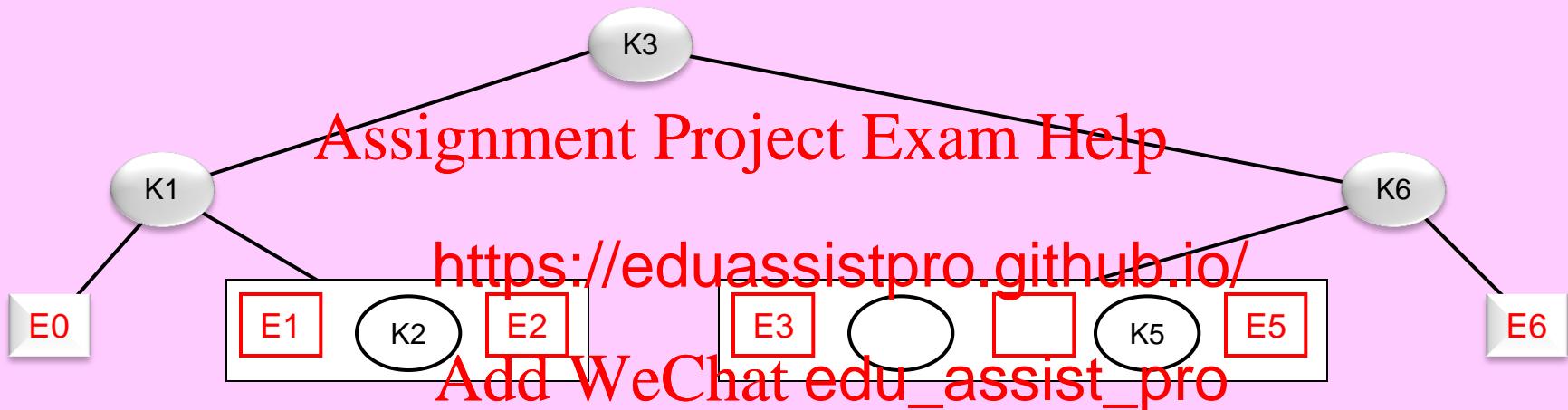
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

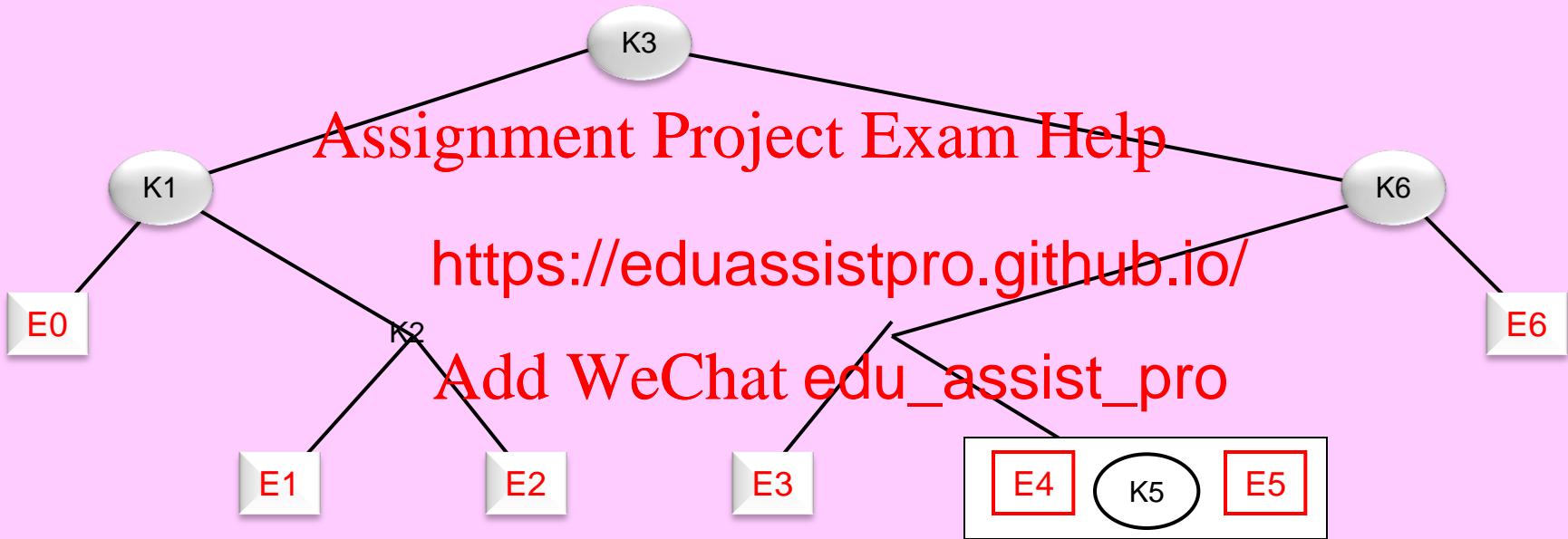
# BST from Binary Search on Sorted Array

E0 < K1 < E1 < K2 < E2 < K3 < E3 < K4 < E4 < K5 < E5 < K6 < E6 < ... < Kn < En



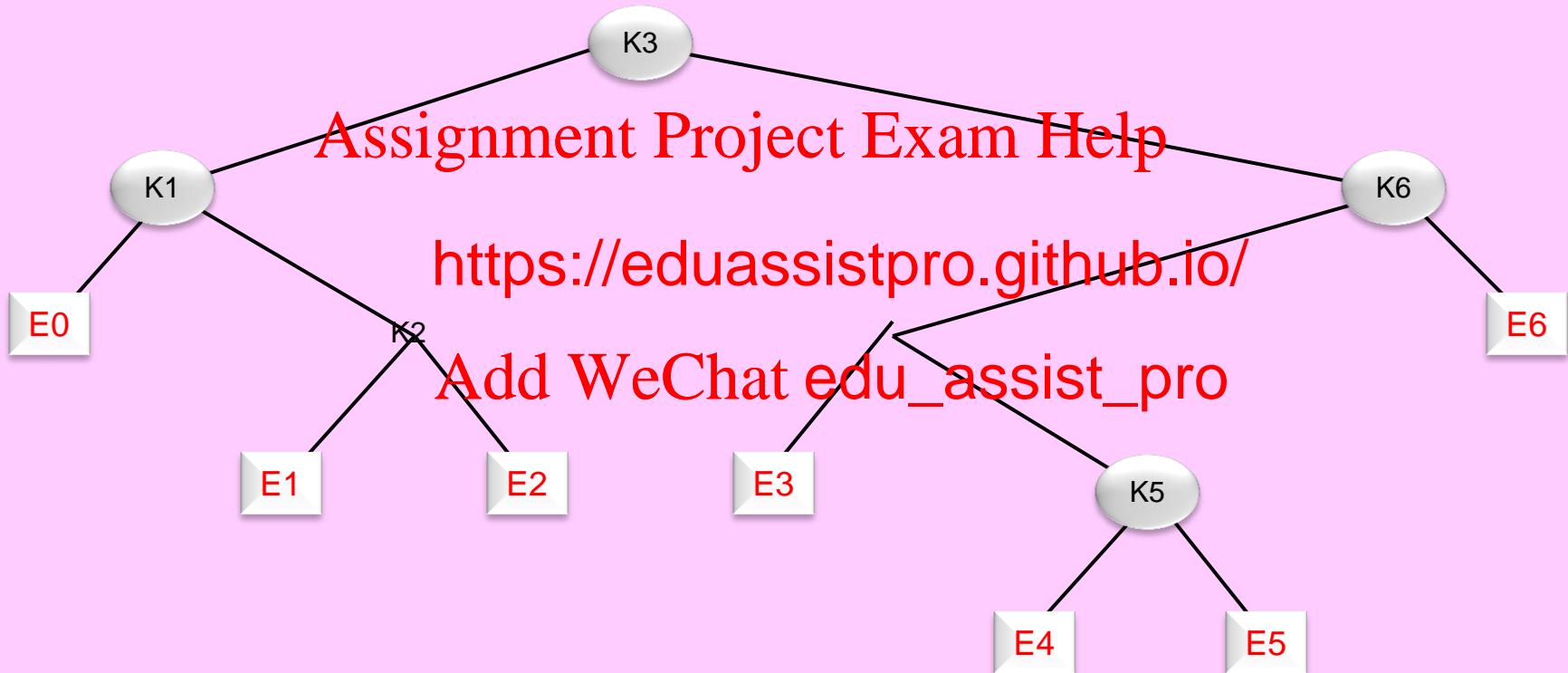
# BST from Binary Search on Sorted Array

E0 < K1 < E1 < K2 < E2 < K3 < E3 < K4 < E4 < K5 < E5 < K6 < E6 < ... < Kn < En



# BST from Binary Search on Sorted Array

E0 < K1 < E1 < K2 < E2 < K3 < E3 < K4 < E4 < K5 < E5 < K6 < E6 < ... < Kn < En



SORTED ORDER    ≡    BST INORDER

# BST Definition

BST is a **binary tree** T with one distinct key per node such that:

- Inorder node sequence of T encounters **keys** in **sorted order**.

- Equivalent definition: For all nodes x & y in T:

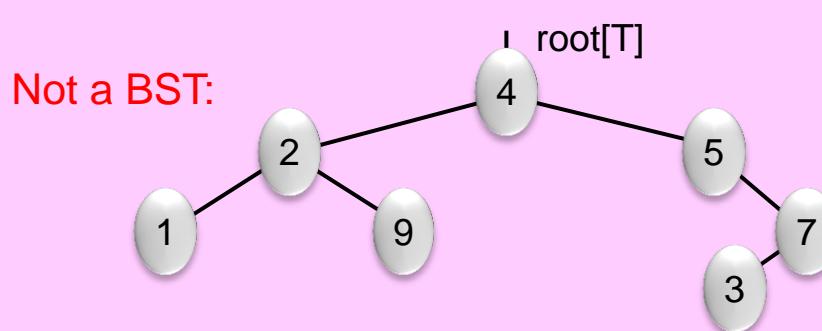
- If x is in the left subtree of y, then  $\text{key}[x] \leq \text{key}[y]$ , and
- If x is in the right subtree of y, then  $\text{key}[x] > \text{key}[y]$ .

<https://eduassistpro.github.io/>

- Wrong definition: For all nodes x & y in T:

- If x is **left child** of y, then  $\text{key}[x] \leq \text{key}[y]$ , and
- If x is **right child** of y, then  $\text{key}[x] > \text{key}[y]$ .

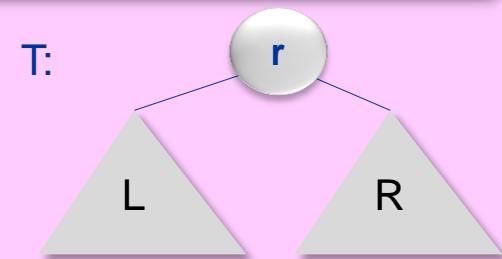
necessary  
but not  
sufficient



# BST Testing

**Input:** A binary tree  $T$  with one key per node.

**Output:** True, if  $T$  is a BST; false otherwise.



**Method 1:** Do an in-order traversal of  $T$  and verify that keys appear in increasing order. This can be done in  $O(n)$  time,  $n = |T|$ .

**Assignment Project Exam Help**

**Observation:**

BSTs have recursive structure <https://eduassistpro.github.io/>

$T$  is a BST  $\Rightarrow$   $L$  and  $R$  are BSTs

Add WeChat edu\_assist\_pro

This implication is necessary but not sufficient.

To make it sufficient, strengthen the conclusion:

$T$  is a BST  $\Leftrightarrow$  (a)  $L$  and  $R$  are BSTs, and  
(b) max key in  $L < \text{key}[r] <$  min key in  $R$

**Method 2:** Do a post-order traversal of  $T$ . How?

# BST Testing

**Input:** A binary tree T with one key per node.

**Output:** True, if T is a BST; false otherwise.

## Algorithm **CorrectBST(T)**

Pre-Condition: T is a binary tree with one key per node

Post-Condition: output is true if T is a BST, false otherwise.

**return** IsBST(root[T])

**end**

<https://eduassistpro.github.io/>

## Function **IsBST(r)**

Pre-Condition: r is pointer to root of a binary tree with one key per node

Post-Condition: output is true if the binary tree is a BST, false otherwise.

Base: **if** r = nil **then return** true

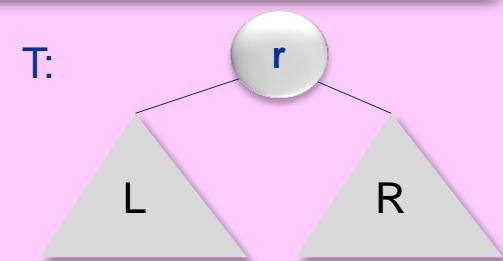
Conquer: SolL  $\leftarrow$  IsBST(left[r]); SolR  $\leftarrow$  IsBST(right[r])

Combine: Sol  $\leftarrow$  MERGE(SolL, SolR) ???

Output: **return** Sol

**end**

Two options: (1) strengthen Pre-Cond or (2) strengthen Post-Cond



# Strengthen Pre- & Post-Condition

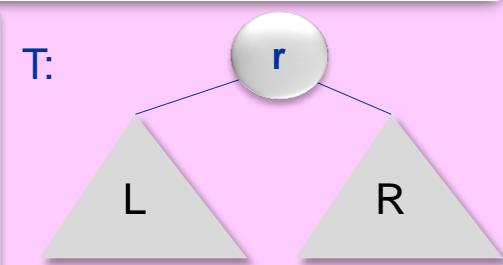
## Algorithm CorrectBST(T)

Pre-Condition: T is a binary tree with one key per node.

Post-Condition: output is true if T is a BST, false otherwise.

```
    return IsBST(root[T], -∞ , +∞ )
```

```
end
```



## Function IsBST(r, minKey, maxKey)

Pre-Condition: r is point one key per node,  
minKey sibly  $\pm\infty$ ).

Post-Condition: output is true if the binary tree d  
minKey < maxKey and for n that tree:  
minKey < key[x] < maxKey; false otherwise.

```
if minKey ≥ maxKey then return false
```

```
if r = nil then return true
```

```
SolL ← IsBST(left[r], minKey, key[r]);
```

```
SolR ← IsBST(right[r], key[r], maxKey)
```

```
Sol ← SolL & SolR
```

```
return Sol
```

```
end
```

# Strengthen Post-Condition

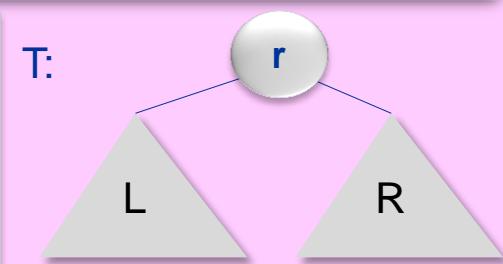
## Algorithm **CorrectBST(T)**

Pre-Condition: T is a binary tree with one key per node

Post-Condition: output is true if T is a BST, false otherwise.

```
(isBST, minKey, maxKey) ← BSTtest(root[T] )  
return isBST
```

end



Assignment Project Exam Help

## Function **BSTtest(r)**

Pre-Condition: r is point to a binary tree with one key per node

Post-Condition: output is

isBST = true if the binary tree is a BST  
isBST  $\Rightarrow$  minKey = minimum key in the tree rooted at r, and  
maxKey = maximum key in the tree rooted at r

```
if r = nil then return (true,  $+\infty$ ,  $-\infty$ )    § Note where  $+\infty$  &  $-\infty$  are  
(isBTL, minL, maxL) ← BSTtest(left[r]);  
(isBTR, minR, maxR) ← BSTtest(right[r])  
isBST ← isBTL & isBTR & (maxL < key[r] < minR)  
return (isBST, min{ minL, key[r] }, max{ maxR, key[r] })
```

end

# Bibliography

## Additional optional sources:

- [Edm08] Jeff Edmonds, "*How to Think about Algorithms*," Cambridge H. Press, 2008.  
[Parts of chapters 1-4, 6-10: good coverage of the loop-invariant method.]

- [AHU74] Aho, Hopcroft, Ullman, "*The Design and Analysis of Computer Algorithms*", Addison-Wesley,  
[A classic text book <https://eduassistpro.github.io/> integer & matrix computations, and more.]

Add WeChat edu\_assist\_pro

- [Man89] Manber, "*Introduction to Algorithms: A Creative Approach*", Addison-Wesley, 1989.  
[Contains interesting induction topics and exercise solutions.]

Assignment Project Exam Help  
  
<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

## INSTRUCTIONS:

In the following problems you are asked to design and analyze efficient iterative or recursive algorithms.

Follow the Loop-Invariant design-&-verification method for iterative algorithms, and the strong induction method for recursive algorithms.

You should clearly state and explain the proof steps as learned in the course.

## Assignment Project Exam Help

### 1. Double Vision in Sorted

Design and analyze an  $O(n \log n)$  algorithm for the following problem.

**Input:** An array  $A[1..n]$

<https://eduassistpro.github.io/>

increasing order.

**Output:** True if there is a pair of indices  $i$  and  $j$  such that  $A[i] < A[j]$ ; false otherwise.

Add WeChat edu\_assist\_pro

### 2. The Longest All-Positive Sub-Array Problem:

**Input:** An array  $A[1..n]$  of arbitrary integers.

**Output:** The longest contiguous sub-array of  $A[1..n]$  with positive entries only.

### 3. The Maximum-Sum Monotone Sub-Array Problem:

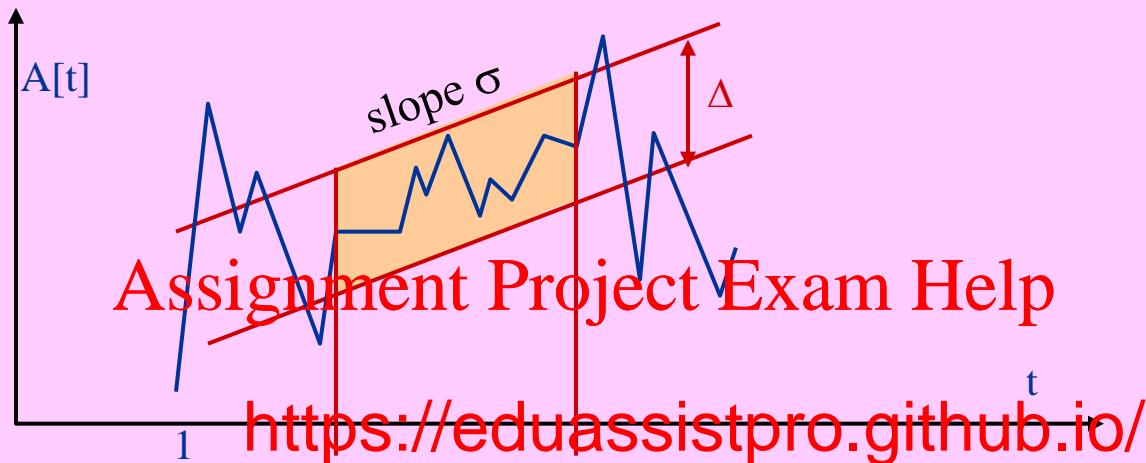
**Input:** An array  $A[1..n]$  of arbitrary positive integers.

**Output:** The maximum-element-sum contiguous sub-array of  $A[1..n]$  whose entries form a monotone sequence (either ascending or descending).

#### 4. The Longest $\Delta$ -Smooth $\sigma$ -Slope Sub-Array Problem:

**Input:** An array  $A[1..n]$  of arbitrary integers, and two numbers  $\sigma$ , and  $\Delta > 0$ .

**Output:** The longest contiguous sub-array, say  $A[i..j]$ , whose entries fall between two lines of slope  $\sigma$  with vertical separation  $\Delta$ . (See the illustrative figure below.)



#### 5. The Largest $\Delta$ -Flat Subset and Sub-Array Problem

Let  $\Delta$  be a positive real number and  $S$  be a finite set of real numbers. We say  $S$  is  **$\Delta$ -flat** if

$$\max(S) - \min(S) \leq \Delta \cdot (|S| - 1).$$

(That is, the average difference between contiguous pairs of elements in the sorted permutation of  $S$  is  $\leq \Delta$ .)

Design and analyze efficient algorithms for the following problems:

- Given a set  $A$  of  $n$  elements and a  $\Delta > 0$ , compute the largest  $\Delta$ -flat subset of  $A$ .
- Given an array  $A[1..n]$  of  $n$  elements and a  $\Delta > 0$ , compute the longest  $\Delta$ -flat contiguous sub-array of  $A$ .

[Note: in part (a) any subset is a potential candidate solution, but in part (b) only those subsets that correspond to contiguous sub-arrays are allowed.]

Hint: for part (b) use an incremental method similar to that of the LSS problem we studied in these slides.]

**6. Design and analyze efficient algorithms for each of the following problems:**

- a) **Input:** An array  $A[1..n]$ , with each element 0 or 1.  
**Output:** the longest contiguous sub-array of  $A[1..n]$  that contains at least as many 1's as 0's.
- b) **Input:** An array  $A[1..n]$ , with each element 0 or 1.  
**Output:** The longest contiguous sub-array of  $A[1..n]$  with at least twice as many 0's as 1's.
- c) **Input:** An array  $A[1..n]$ , with each element 0 or 1.  
**Output:** The longest contiguous sub-array of  $A[1..n]$  such that the number of 0's minus the number of 1's in that sub-array is  $2 \bmod 5$ .
- d) **Input:** An array  $A[1..n]$ , with each element 0 or 1.  
**Output:** The Length of longest monotone ascending (not necessarily contiguous) sub-array of  $A[1..n]$ .
- e) **Input:** An array  $A[1..n]$  of integers in strictly increasing order.  
**Output:** Find an index  $i$  such that  $A[i] = \frac{n+1}{2}$ .  
ex exists.
- f) **Input:** An array  $A[1..n]$  where  $1 \leq A[i] \leq 5$  and sum of elements is non-negative.**Output:** Longest contiguous subarray of  $A[1..n]$  such that the sum of elements is non-negative.

**Assignment Project Exam Help**

- 7. Inplace Rank Sort:** We are given an array  $A[1..n]$  of  $n$  integers. Each employee record  $A[i]$  consists of two fields:  $A[i].in$  and  $A[i].out$ . The rank of each employee is one of 4 possibilities  $\{1,2,3,4\}$ . Design and analyze a linear-time, in-place incremental algorithm to sort the  $n$  employee records in ascending order of rank.

- 8. Generalize 3-Partition to C-Partition:** For any fixed integer  $C$ ,  $2 \leq C \leq n$ , show that the  $C$ -partition problem can be solved in  $O(Cn)$  time, using  $O(C)$  extra memory cells.

- 9. 2SUM & 3SUM in a Sorted Array:** We are given a sorted array  $A[1..n]$  and a target value  $X$ .

- a) 2SUM: Design & analyze an efficient algorithm to find a pair of array indices  $(i,j)$  such that  $A[i] + A[j] = X$ , or report that no such pair exists. [Hint: can be done in  $O(n)$  time.]
- b) 3SUM: Similar to part (a), find a triple array indices  $(i,j,k)$  such that  $A[i] + A[j] + A[k] = X$ . What is the time complexity of your algorithm?

- 10. A Fibonacci Identity:** Let n and m be arbitrary natural numbers.
- (a) Prove the identity below by induction on n. (Make sure to cover all required base cases.)
- (b) Prove this identity using the matrix formula  $G(n) = A^n$  we established on page 70 of these slides.
- $$F_{n+m+1} = F_{n+1}F_{m+1} + F_nF_m$$
- 11. Euclid & Fibonacci:** Prove that GCD of two Fibonacci numbers is a Fibonacci number.
- 12. Integer Linear Equations:** Design and analyze an efficient algorithm that, given integers a, b, c, determines whether the linear equation  $ax + by = c$  has an integer solution for variables x and y, and if so it finds one such solution.
- Assignment Project Exam Help**
- 13. Modified Fibonacci Sequence:** Define  $G_0 = 0$ ,  $G_1 = 1$ ,  $G_2 = \dots$  for  $n \geq 0$ .  
Design and analyze an  $O(\log n)$  time algorithm to compute  $G_n$ .
- Add WeChat edu\_assist\_pro**
- 14. Closest Red-Blue Pair:** We are given a set of n points in the plane, each point is either red or blue. We want to find the closest (red, blue) pair among the input points (if there is any such pair). Design and analyze an efficient divide-&-conquer algorithm for this problem.
- 15. Boys & Girls in a Binary Search Tree (BST):**  
Each node item in our Binary Search Tree is a record of the form (gender, salary), where gender  $\in \{\text{boy, girl}\}$ , and salary is a positive integer which is used as the key. Design & analyze an efficient recursive algorithm to find a pair of opposite sex records with closest salary values.

## 16. Strengthening Recursion Pre/Post Conditions:

**Below we show two algorithms for the following problem:**

Given a binary tree T and a height threshold h, return the number of nodes in T of height at least h.

**Algorithm CountHighNodes1(T, h)**

Count  $\leftarrow$  CHN1(root[T], h)

**return** (Count)

**end**

**Function** CHN1(r, h)

**if** r = nil **then return** (0)

LCount  $\leftarrow$  CHN1(left[r], h)

RCount  $\leftarrow$  CHN1(right[r], h)

Height  $\leftarrow$  ComputeHeight(r)

Count  $\leftarrow$  LCount +

**if** Height  $\geq$  h **then C**

**return** (Count)

**end**

**Function** ComputeHeight(r)

**if** r = nil **then return** (-1)

LHeight  $\leftarrow$  ComputeHeight(left[r])

RHeight  $\leftarrow$  ComputeHeight(right[r])

**return** (1 + max(LHeight, RHeight))

**end**

**Algorithm CountHighNodes2(T, h)**

(Count, RootHeight)  $\leftarrow$  CHN2(root[T], h)

**return** (Count)

**end**

**Function** CHN2(r, h)

**if** r = nil **then return** (0, -1)

t, LHeight)  $\leftarrow$  CHN2(left[r], h)

t, RHeight)  $\leftarrow$  CHN2(right[r], h)

$\leftarrow$  1 + max(LHeight, RHeight)

Count + RCount

**h then Count  $\leftarrow$  Count + 1**

unit, Height)

**end**

- a) Give all the Pre- and Post-Conditions, and argue that both algorithms CountHighNodes1 and CountHighNodes2 correctly solve the problem.
- b) Analyze the worst-case running times of CountHighNodes1 and CountHighNodes2.
- c) Explain the real reason why CountHighNodes1 is less efficient than CountHighNodes2.

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro

## 17. More Recursion on Binary Trees:

The input is a binary tree  $T$  with an integer valued key per node.

Design & analyze efficient recursive algorithms for the following tasks.

[If no solution exists, some special value, say null, may be returned.  
If there are several solutions, choose any one of them and return it.]

- a) Find the sum of the keys in all the leaves of  $T$ .
- b) Find a root-to-leaf path with minimum sum key sequence.
- c) Find the root of the largest subtree that contains no more than  $|T|/10$  nodes.  
[Note:  $|T|$  denotes the number of nodes in  $T$  and is not part of the input.]
- d) Find a node with maximum difference between the heights of its children.  
[Note: height of a node is the difference between the heights of its children. Height of an empty tree is  $-1$ .]
- e) Find a pair of nodes  $(x, y)$  in  $T$  such that
  - (i)  $y$  is the successor of  $x$  in the inorder sequence of  $T$ , and
  - (ii)  $|(\text{depth of } x) - (\text{depth of } y)|$  is maximum possible.
- f) Find a pair of nodes  $(x, y)$  in  $T$  with longest **separation**, where the separation between two nodes is the length (number of edges) of the unique path from  $x$  to  $y$  when thinking of the tree as an undirected graph, i.e., from  $x$  up to the lowest common ancestor and back down to  $y$ .

[Hint: Look up the previous exercise. Sample Midterm Solutions contain additional examples.  
You may need to strengthen the Pre- or Post-Conditions of your recursive routines.  
For instance, let the recursive routine return extra info in addition to the required output data.  
Then use the returned extra info in the “combine” step of the recursion.]

## More Recursion on Binary Trees:

### 18. The Most Weird Species:

**Input:** A binary tree T in which each node x contains a field  $color[x]$  which is either *red* or *blue*.

Define **lineage** of a node x to be the set of all those nodes in T that are either ancestor or descendant of x (including x). Define **weirdness** of a node x to be the difference between the number of red nodes and the number of blue nodes in the lineage of x.

**Output:** A node x in T with maximum weirdness.

### 19. Richest Heritage: Assignment Project Exam Help

**Input:** A binary tree T in which each node x contains a field  $th[x]$ , which is a (positive, zero, or negative) monetary value.

<https://eduassistpro.github.io/>

Define (monetary) **heritage** of a node x to be the sum of the monetary values of its descendants of x minus the monetary value of x itself.

[Add WeChat edu\\_assist\\_pro](#)

**Output:** A node x in T with maximum heritage.

### 20. Saddle Point:

**Input:** A binary tree T in which each node x contains a field  $value[x]$  which is a real number.

We say a node x in T is a **saddle point** if x has minimum value among all its ancestors (including x), but it has maximum value among all its descendants (including x).

**Output:** A saddle point of T if there is one, or null if there isn't.

## 21. Binary Tree Bisection:

Many divide-&-conquer algorithms that operate on graphs require that the graph be bisected into nearly equal-sized subgraphs by removing a small number of edges. We will discuss graph algorithms later. This problem investigates bisection of binary trees.

- a) Prove the following Fact.

**Fact:**  $\forall n \geq 2$ : any  $n$ -node binary tree  $T$  has a **balanced edge**: an edge whose removal will separate  $T$  into two binary trees, each having at most  $\lceil (2n-1)/3 \rceil$  nodes.

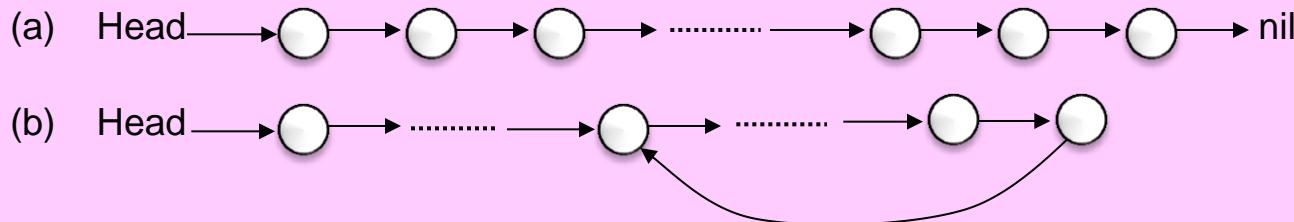
Assignment Project Exam Help

- b) Design and analyze an algorithm to partition the nodes of any given binary tree  $T$ .  
[Hint:  $O(n)$  time is possible.]
- c) Show that the bound in the above Fact is optimal by demonstrating that for all  $n \geq 2$  there is an  $n$ -node binary tree whose Add WeChat [edu\\_assist\\_pro](https://eduassistpro.github.io/) single edge results in one of its two parts having at most  $\lceil (2n-1)/3 \rceil$  nodes.
- d) **Binary Tree Bisection:** Using part (a), show that by removing at most  $O(\log n)$  edges, we can partition the nodes of any  $n$ -node binary tree into two sets  $A$  and  $B$  such that  $|A| = \lceil n/2 \rceil$  and  $|B| = \lfloor n/2 \rfloor$ , and there is no remaining edge with one end vertex in  $A$  and the other in  $B$ . For this part you don't need to give a detailed algorithm. A high level description will suffice.

## 22. Linked-List Correctness Verification (an IBM interview test question):

We are given the head pointer to a read-only linked list. Each node of this list consists of a single field, namely, a pointer to the next node on the list. For the list to have correct structure, each node should point to its successor, except for the last node that points to *nil*, as in figure (a) below. However, due to a possible structural error, the list may look like that of figure (b), where a node points to one of the previous nodes (possibly itself). From that node on, we no longer have access to the rest of the nodes on the list. The task is to verify the structural correctness of the list. If we knew  $n$ , the number of accessible nodes on the list, we could easily verify the list's structural correctness, namely, starting from the head pointer, we would follow the list nodes for  $n$  steps and see if we reach a *nil* pointer. However, we have no advance knowledge of the length of the list.

Design & analyze a linear-time in-place algorithm to test the structural correctness of the list. That is, your algorithm should take time linear in the number of accessible nodes, and work in-place, i.e., use only  $O(1)$  scalar variables, but not su <https://eduassistpro.github.io/> not have any additional fields. [Hint: use repeated doubling.] Add WeChat **edu\_assist\_pro**

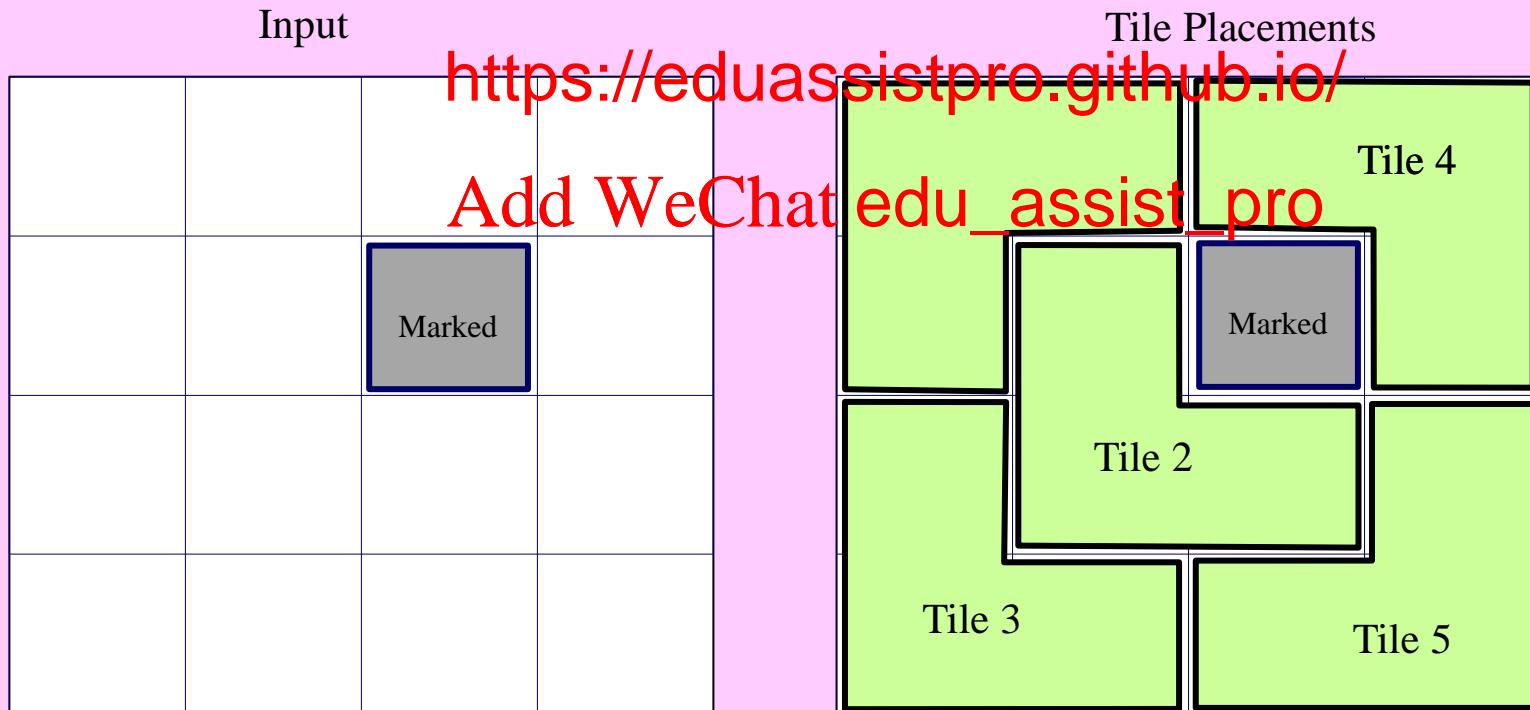


**23. A Tiling Problem:** The input is a  $2^n$  by  $2^n$  square checkerboard with the unit square at the given coordinates (i,j) marked.

You have a sufficient number of identical tiles of the shape  that can cover 3 board squares.

Your task is to place non-overlapping tiles on the board to cover each of the  $2^n \times 2^n$  board squares except the marked one. Give a recursive algorithm for this problem.

**Further explanation:** You need to output the list of tile placements. A tile can cover 3 neighboring squares as mentioned above. So, each tile placement can be given as the set of those 3 squares; each square will be specified by its coordinates. See the example below:  
**Assignment Project Exam Help**



**24. [Goodrich-Tamassia C-1.14. pp. 50-53]**

Suppose you are given a set of small boxes, numbered 1 to  $n$ , identical in every respect except that each of the first  $i$  contain a pearl whereas the remaining  $n - i$  are empty. You also have two magic wands that can each test if a box is empty or not in a single touch, except that a wand disappears if you test it on an empty box. Show that, without knowing the value of  $i$ , you can use the two wands to determine all the boxes containing pearls using at most  $O(n)$  wand touches. Express, as a function of  $n$ , the asymptotic number of wand touches needed.

**25. [Goodrich-Tamassia C-1.25. pp. 50-53]**

An evil king has a cellar containing  $n$  bottles of expensive wine, and his guards have just caught a spy trying to poison the king's wine. Fortunately, the guards caught the spy after he succeeded in poisoning only one bottle. Unfortunately, they don't know which one. To make matters worse, the poison the spy used was very deadly; just one drop diluted even a billion to one will still kill someone. It takes a full month for the person to die. Design a algorithm to find exactly which one of his wine bottles was poisoned in at most  $O(\log n)$  of his taste testers.

Add WeChat [edu\\_assist\\_pro](https://eduassistpro.github.io/)

**26. [CLRS 2<sup>nd</sup> edition Problem 4-2, p. 85] Finding the missing integer:**

An array  $A[1..n]$  contains all the integers from 0 to  $n$  except one. It would be easy to determine the missing integer in  $O(n)$  time by using an auxiliary array  $B[0..n]$  to record which numbers appear in  $A$ . In this problem, however, we cannot access an entire integer in  $A$  with a single operation. The elements of  $A$  are represented in binary, and the only operation we can use to access them is "*fetch the  $j^{th}$  bit of  $A[i]$* ," which takes constant time. Show that if we use only this operation, we can still determine the missing integer in  $O(n)$  time.

## 27. Polynomial multiplication by divide-&-conquer:

A degree  $n-1$  polynomial  $P(x) = \sum_{i=0}^{n-1} p_i x^i = p_0 + p_1 x + p_2 x^2 \dots + p_{n-1} x^{n-1}$  can be represented by an array  $p[0..n - 1]$  of its  $n$  coefficients. Suppose  $P(x)$  and  $Q(x)$  are two polynomials of degree  $n-1$ , each given by its coefficient array representation. Their product  $P(x)Q(x)$  is a polynomial of degree  $2(n-1)$ , and hence can be represented by its coefficient array of length  $2n-1$ . The polynomial multiplication problem is to compute the coefficient array of  $P(x)Q(x)$ , given the coefficient arrays of  $P(x)$  and of  $Q(x)$ .

There is an obvious  $\Theta(n^2)$  (i.e., quadratic) time algorithm to solve this problem. However, a method similar to the divide-&-conquer integer multiplication algorithm of Karatsuba-Ofman can achieve sub-quadratic time complexity. Design and analyze one such sub-quadratic algorithm for the polynomial multiplication problem.

**Assignment Project Exam Help**

<https://eduassistpro.github.io/>

## 28. [CLRS Problem 30-2, p. 921] Toeplitz Matrix

A **Toeplitz matrix** is an  $n \times n$  matrix  $A = (a_{ij})_{i=1, j=1}^{i=n, j=n}$  for  $i = 2..n$  and  $j = 2..n$ .

- a) Is the sum of two Toeplitz matrices necessarily Toeplitz? What about the product?
- b) Describe how to represent a Toeplitz matrix so that you can add two  $n \times n$  Toeplitz matrices in  $O(n)$  time.
- c) Give an  $O(n \log n)$  time divide-&-conquer algorithm for multiplying an  $n \times n$  Toeplitz matrix by a vector of length  $n$ . Use your representation from part (b).  
[Hint: you may need to learn how the Fast Fourier Transform (FFT) algorithm works.]
- d) Give an efficient algorithm for multiplying two Toeplitz matrices. Analyze its running time.

- 29. Particle Physics Problem:** You have been working with some physicists who need to study, as part of their experimental design, the interaction among large numbers of very small charged particles. Basically, their setup works as follows. They have an inert lattice structure, and they use this for placing charged particles at regular spacing along a straight line. Thus we can model their structure as consisting of the points  $\{1, 2, 3, \dots, n\}$  on the real line; and at each of these points  $j$ , they have a particle with charge  $q_j$ . (Each charge can be either positive or negative.)

They want to study the total force on each particle, by measuring it and then comparing it to a computational prediction. This computational predictive part is where they need your help. The total net force on particle  $j$ , by Coulomb's Law, is equal to

$$F_j = \sum_{i:i < j} \frac{C q_i q_j}{(j-i)^2} + \sum_{i:i > j} \frac{C q_i q_j}{(j-i)^2}.$$

They have written the following pseudocode for all  $j$ .

**for**  $j \leftarrow 1..n$  **do**

$F_j \leftarrow 0$

**for**  $i \leftarrow 1..j-1$  **do**

$F_j \leftarrow F_j +$

$E_j - E_i$

**for**  $i \leftarrow j+1 .. n$  **do**

$F_j \leftarrow F_j - E_j + E_i$

Add WeChat edu\_assist\_pro

**output**  $F_j$

**end**

It is not hard to see that this algorithm takes  $O(n^2)$  time.

The trouble is, for the large values of  $n$  they are working with, the program takes several minutes to run. On the other hand, their experimental setup is optimized so that they can throw down  $n$  particles, perform the measurements, and be ready to handle  $n$  more particles within a few seconds. So they would really like it if there were a way to compute all the forces  $F_j$  much more quickly, so as to keep up with the rate of the experiment.

Help them out by designing a divide-&-conquer algorithm that computes all the forces  $F_j$  in  $O(n \log n)$  time. [Hint: use part (c) of the previous problem.]

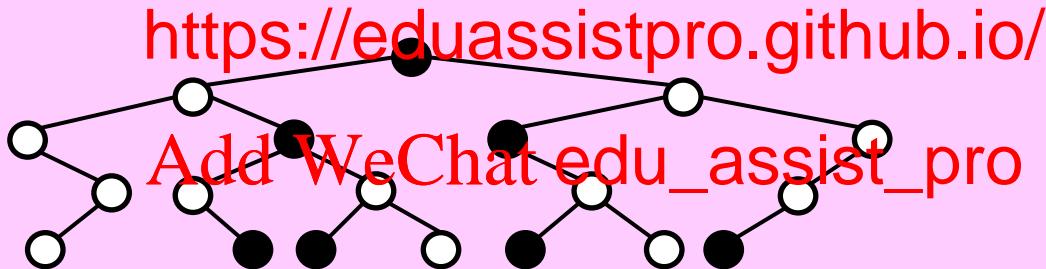
**30. Input:** A binary tree  $T$  in which each node  $x$  contains a real valued field named  $value[x]$ . We say a node  $x$  in  $T$  is **value-balanced** if and only if the average value of ancestors of  $x$  (including  $x$ ) equals the average value of descendants of  $x$  (including  $x$ ).

**Output:** Among value-balanced nodes in  $T$  return the one with minimum depth (break ties arbitrarily). Return  $nil$  if  $T$  has no value-balanced node.

**Note:** Apart from the value field and left child and right child pointers, nodes of  $T$  do not have any additional storage fields.

Design and analyze an efficient algorithm for this.

**31.** A node  $x$  in a binary tree is said to be **well-oriented** if the path from root to  $x$  follows an equal number of left child branches as right child branches. For example, in the binary tree below, all the well-oriented nodes



You may assume that for a node  $x$ ,  $left[x]$  is left child of  $x$ ,  $right[x]$  is right child of  $x$ . There are no other storage fields in the nodes of the tree.

Design and analyze an efficient algorithm that returns the number of well-oriented nodes in a given binary tree  $T$ .

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu\_assist\_pro