Introduction to Algorithms

Meng-Tsung Tsai

09/10/2019

Course Materials

<u>Textbook</u>

Introduction to Algorithms (I2A) 3rd ed. by Cormen, Leiserson, Rivest, and Stein.

<u>Reference Book</u>

Algorithms (JfA) 1st ed. by Erickson. An e-copy can be downloaded from author's website: http://jeffe.cs.illinois.edu/teaching/algorithms/

<u>Websites</u>

http://e3new.nctu.edu.tw for slides, written assignments, and solutions.

http://oj.nctu.me for programming assignments.

Office Hours

<u>Lecturer's</u>

On Wednesdays 16:30 - 17:20 at EC 336.

TA. Erh-Hsuan Lu (呂爾軒) and Tsung-Ta Wu (吳宗達)

On Mondays 10:10 - 11:00 at ES 724.

More TA hours will be announced.

Grading Policy

1. No plagarism and cheating. You may fail this course by doing this.

2. Saying I don't know is better than talking nonsense. In written assignments and quizzes, the midterm exam, and the final exam, you receive 25% credits if you explictly write down "I don't know." Leaving blank or talking nonsense gives you 0 point.

3. Your final grade will be at least

$$\mathrm{Min}(\,(2\mathrm{Max}(A, B) + \mathrm{Min}(A, B) + C + D)/5,\ 99)$$

where A is the average of your written assignments and quizzes, B is the average of your programming assignments and quizzes, and C (resp. D) denotes your grade of the midterm exam (resp. the final exam).

4. Anyone who fails this course may ask for a make-up exam only if he/she participates in class regularly.

Important Dates

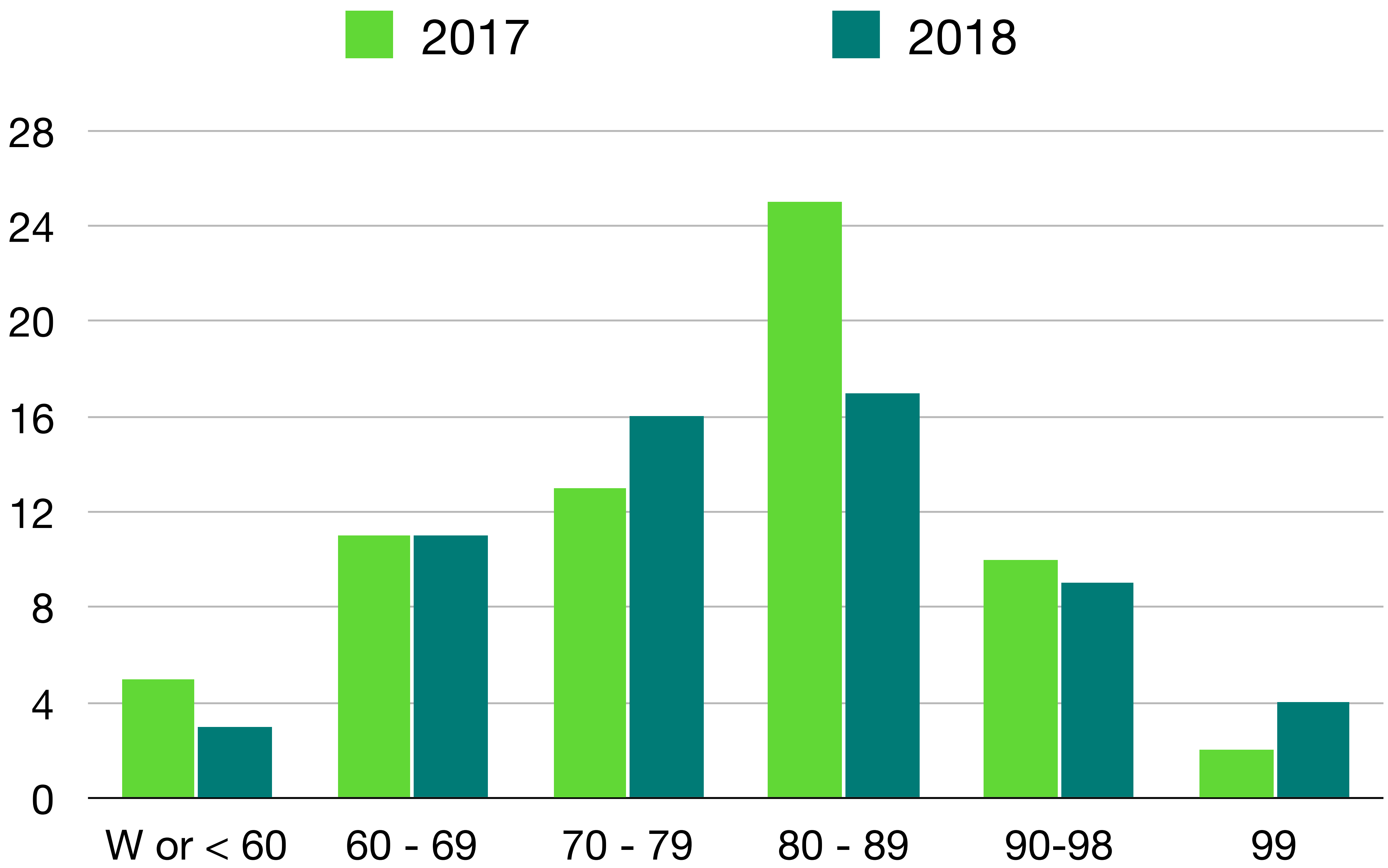
<u>In Class</u>

Oct 24: Quiz 1
Nov 05: Midterm Exam
Dec 31: Quiz 2
Jan 07: Final Exam

<u>Outside of Class</u>

Nov 16 (Sat 13:30 - 17:30): Programmign Quiz 1
Dec 28 (Sat 13:30 - 17:30): Programming Quiz 2

Distribution of Grades



What are algorithms?

What is an algorithm?

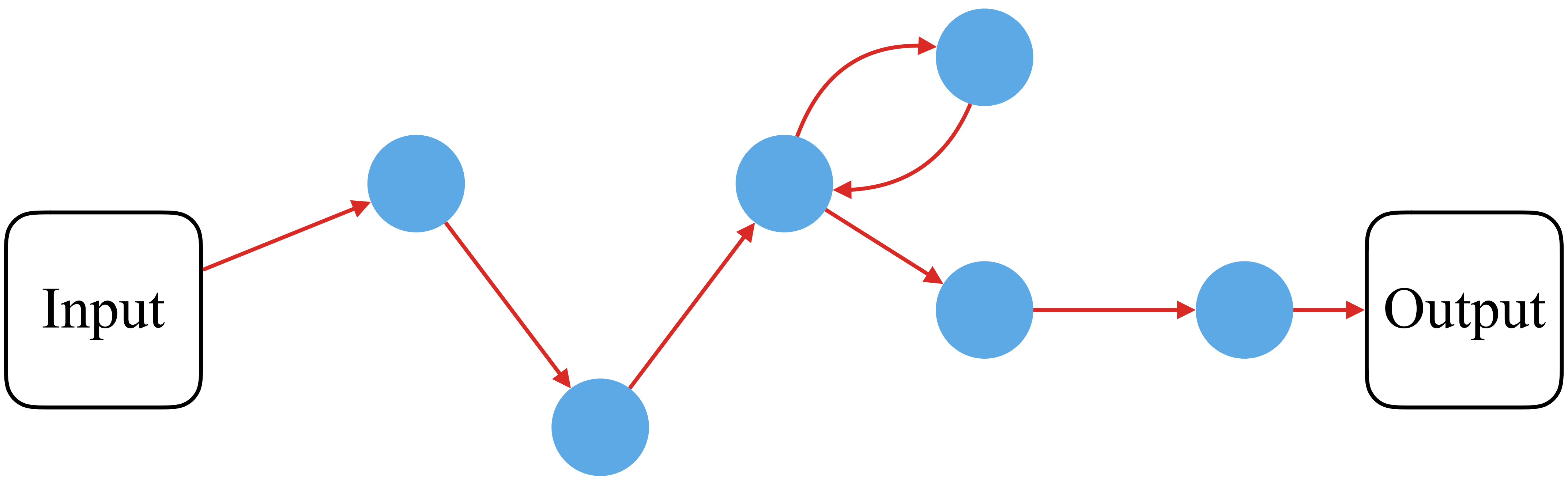
<u>Formally</u>, given the specification of a problem

What is an algorithm?

Formally, given the specification of a problem

What is an algorithm?

Formally, given the specification of a problem



an algorithm is computational procedures that take some values as input and produce some values as output.

What is an algorithm?

Here is an <u>informal</u> example:

Input

Output

What is an algorithm?

Here is an informal example:

Algorithm 1

foreach egg{

}

Input

Output

What is an algorithm?

Here is an <u>informal</u> example:

Input

Algorithm 1

foreach egg{

}

Algorithm 2

foreach egg{

}

Output

Selection Sort

The Champion Problem

Input: an array A of n integers.

Output: an index k so that A[k] is the minimum value in A.

A problem instance (an instance)

4 6 2 7 5 1 8 3

return value (ret): null

The Champion Problem

Input: an array A of n integers.

Output: an index k so that A[k] is the minimum value in A.

A problem instance (an instance)

return value (ret): 

The Champion Problem

Input: an array A of n integers.

Output: an index k so that A[k] is the minimum value in A.

A problem instance (an instance)

return value (ret):

The Champion Problem

Input: an array A of n integers.

Output: an index k so that A[k] is the minimum value in A.

A problem instance (an instance)

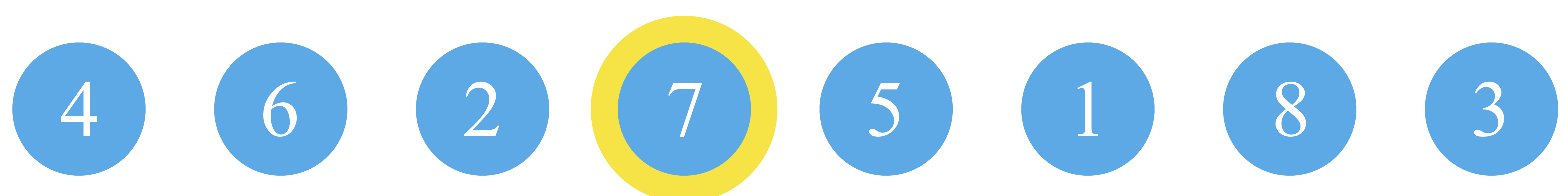
return value (ret):

The Champion Problem

Input: an array A of n integers.

Output: an index k so that A[k] is the minimum value in A.

A problem instance (an instance)



return value (ret):

The Champion Problem

Input: an array A of n integers.

Output: an index k so that A[k] is the minimum value in A.

A problem instance (an instance)

return value (ret):

The Champion Problem

Input: an array A of n integers.

Output: an index k so that A[k] is the minimum value in A.

A problem instance (an instance)

return value (ret):

The Champion Problem

Input: an array A of n integers.

Output: an index k so that A[k] is the minimum value in A.

A problem instance (an instance)

return value (ret):

The Champion Problem

Input: an array A of n integers.

Output: an index k so that A[k] is the minimum value in A.

A problem instance (an instance)

return value (ret):

C++ Code

```
int champion(int *s, int n){ // return -1 for empty input

    int ret = -1; // 1 assignment

    for(int i=0; i<n; ++i){ // incur 2n comparisons, ≤ n-1 assignments,
        if(s[i] < s[ret]){       // and n increments
            ret = i;
        }
    }
    return ret;
} // a constant number of operations for the overhead of function call
```

--- total running time ---

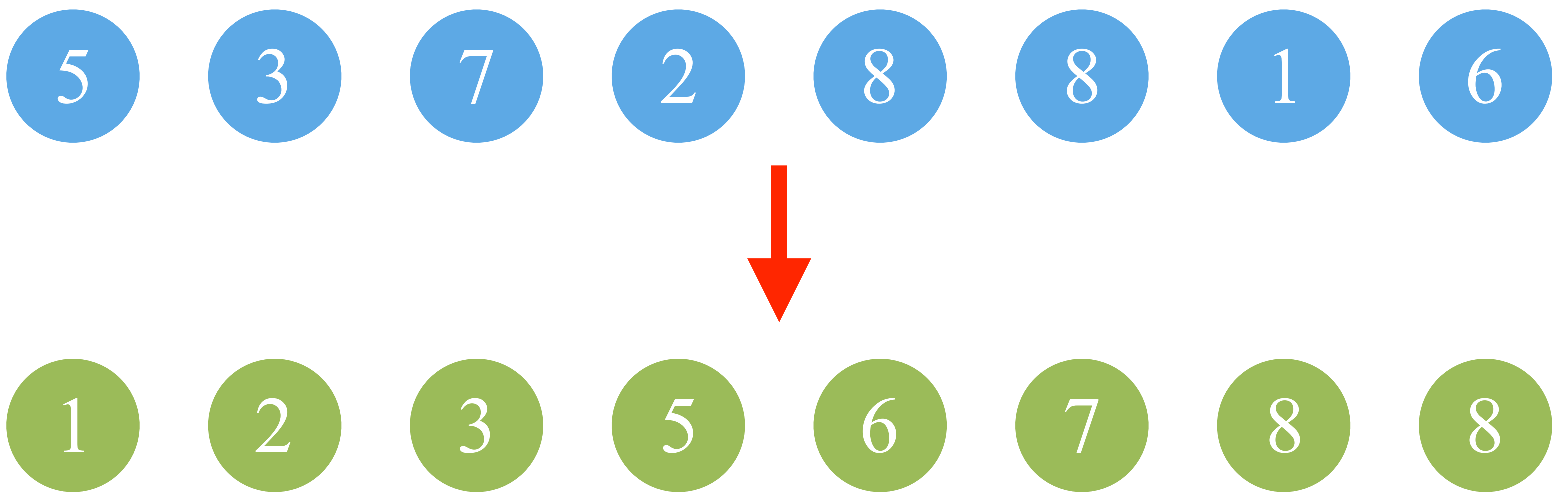
champion() uses at most $4n + C$ operations for some constant C.

Sorting Problem

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance

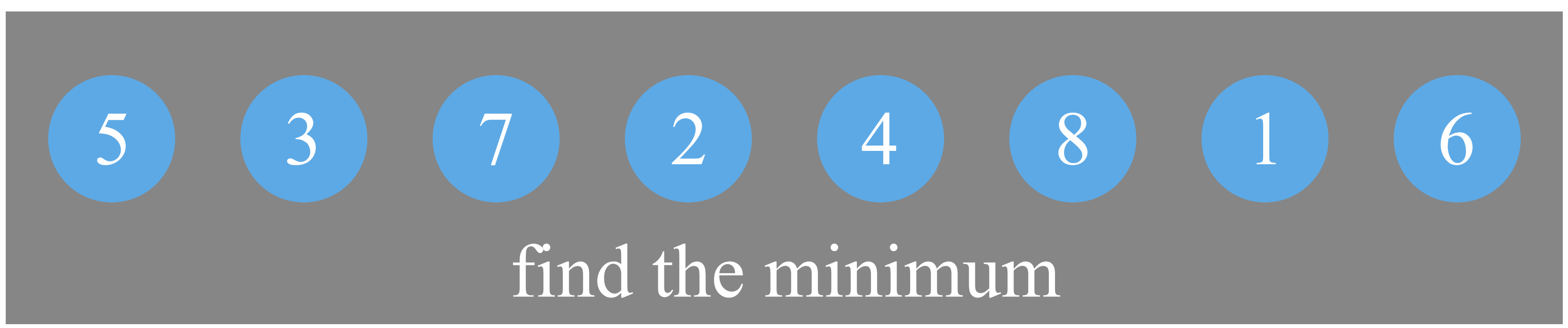


Selection Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance

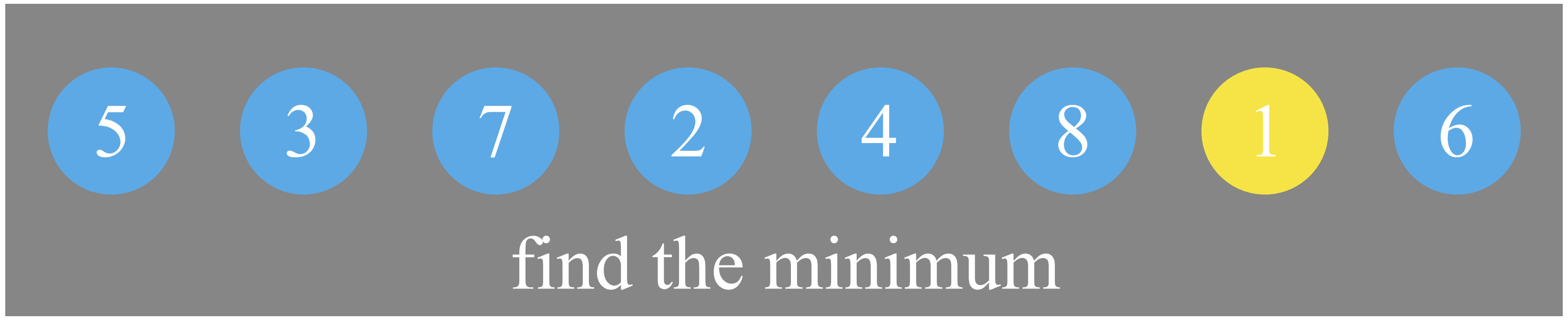


Selection Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance



Selection Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance

Selection Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance

Selection Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance

Selection Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance

Selection Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance

C++ Code

```
void selection_sort(int *s, int n){

    for(int i=0; i<n; ++i){
        int k = champion(s+i, n-i);
        int swap = s[i]; s[i] = s[k]; s[k] = swap;
    }
}
```

--- about the highlight ---

It is called *reduction*. Reducing one problem X to another problem Y means to devise an algorithm for X using an algorithm for Y as a building block.

selection_sort() uses at most n(4n+C+3) operations for some constant C.

C++ Code

```
void selection_sort(int *s, int n){

    for(int i=0; i<n; ++i){
        int k = champion(s+i, n-i);
        int swap = s[i]; s[i] = s[k]; s[k] = swap;
    }
}
---------
```

selection_sort() uses at most $n(4n+C+3)$ operations for some constant C.

Why does the count of operations matter?

C++ Code

```
void selection_sort(int *s, int n){

    for(int i=0; i<n; ++i){
        int k = champion(s+i, n-i);
        int swap = s[i]; s[i] = s[k]; s[k] = swap;
    }
}
---------
```

selection_sort() uses at most $n(4n+C+3)$ operations for some constant C.

Why does the count of operations matter?

A: We can use it to estimate the running time of the program. 10^8 operations takes roughly 1 second. Hence, sorting 10^4 integers by selection sort takes roughly 4 seconds.

C++ Code

```
void selection_sort(int *s, int n){

    for(int i=0; i<n; ++i){
        int k = champion(s+i, n-i);
        int swap = s[i]; s[i] = s[k]; s[k] = swap;
    }
}
```

--- about the highlight ---

Can we replace the highlighted part with s[i] ^= s[k] ^= s[i] ^= s[k]?

C++ Code

```
void selection_sort(int *s, int n){

    for(int i=0; i<n; ++i){
        int k = champion(s+i, n-i);
        int swap = s[i]; s[i] = s[k]; s[k] = swap;
    }
}
```

--- about the highlight ---

Can we replace the highlighted part with s[i] ^= s[k] ^= s[i] ^= s[k]?

No. Why?

Insertion Sort

Insert x into a sorted array A while retaining A sorted

Input: a sorted array A of n integers and an integer x.

Output: a sorted array that comprises all elements in A and x.

An instance

Insert x into a sorted array A while retaining A sorted

Input: a sorted array A of n integers and an integer x.

Output: a sorted array that comprises all elements in A and x.

An instance

Insert x into a sorted array A while retaining A sorted

Input: a sorted array A of n integers and an integer x.

Output: a sorted array that comprises all elements in A and x.

An instance

Insert x into a sorted array A while retaining A sorted

Input: a sorted array A of n integers and an integer x.

Output: a sorted array that comprises all elements in A and x.

An instance

C++ Code

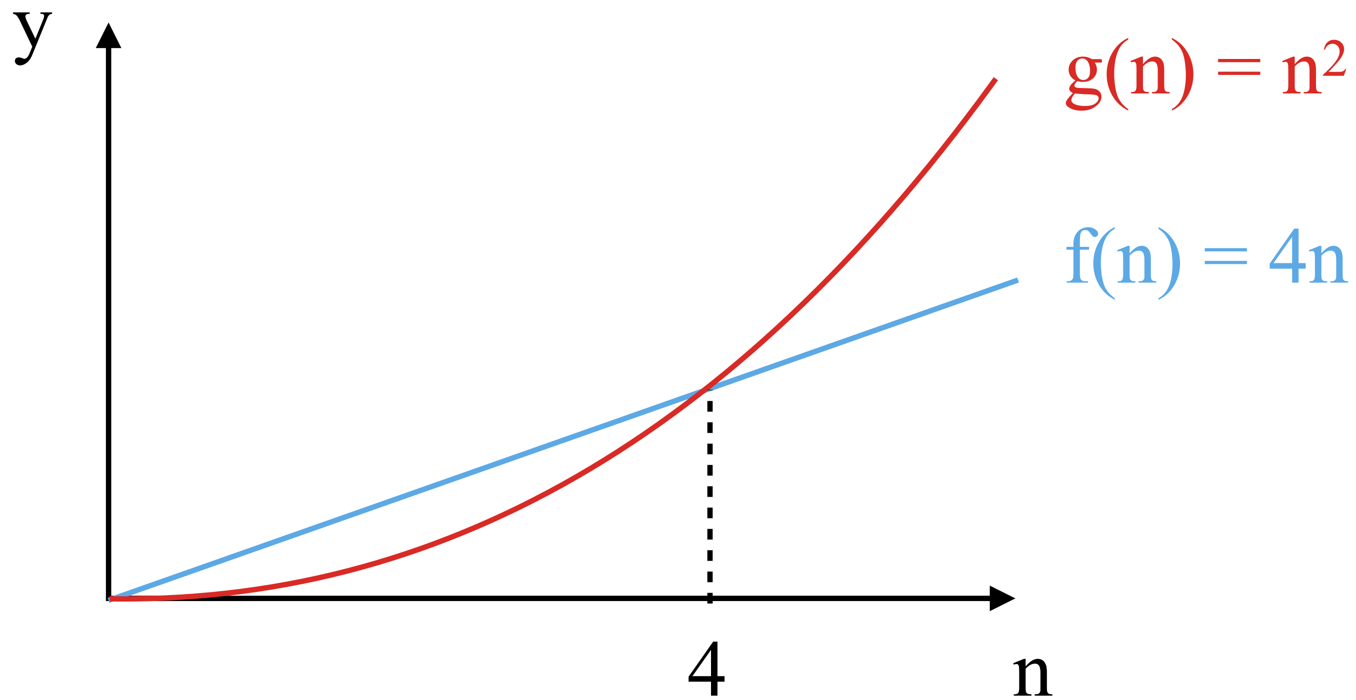
```
void insert(int *s, int n, int x){ // array s has length ≥ n+1
0:    bool placed = false; // whether x has been placed in s
1:    for(int i=n-1; i>=0 && !placed; --i){
2:       if(s[i] > x){
3:            s[i+1] = s[i];
4:       }else{
5:            s[i+1] = x; placed = true;
6:       }
7:    }
8:    if(!placed) s[0] = x;
9:}
```

Line 1 comprises 1 assignment, n comparisons, and n decrements.
Line 2 comprises n comprisons and n dereference.
Line 3 comprises n assignments, n additions, and 2n dereferences ...
It is curbersome (and error-prone) to count the exact operations that an algorithm uses.

Asymptotic Notation: O-Notation

O(g(n)) is pronounced as big-Oh of g of n.

$f(n) = O(g(n))$ means that $f(n) \leq C \cdot g(n)$ for every $n \geq n_0$ for some constants C and n_0.



We can write $4n = O(n)$ by setting $(C, n_0) = (4, 1)$ or $4n = O(n^2)$ by setting $(C, n_0) = (1, 4)$.

Asymptotic Notation: O-Notation

O(g(n)) is pronounced as big-Oh of g of n.

More formally, $f(n) = O(g(n))$ means that $f(n)$ is a function contained in thet set of functions $\{h(n)$: there exists positive constants n_0 and C so that $C \cdot g(n) \geq h(n)$ for every $n \geq n_0\}$.

Because it is curbersome to determine the constant C and we simply need to estimate the running time, we usually use asymptotic notation to denote the time complexity of an algorithm.

--- Example ---

1. Selection sort runs in $O(n^2)$ time, so it can sort 10^4 integers in seconds.

2. Insert *x* into a sorted array runs in $O(n)$, so in seconds one can complete 10^4 insertions.

Exercises

1. $2n^2 + 100\, n - 2000 = O(n^2)$?

2. $2n^3 - 100\, n^2 = O(n^2)$?

3. $n^n = O(n!)$?

4. $n \log n = O(n^{1.5})$?

5. $\log n! = O(n \log n)$?

Insertion Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance

Insertion Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

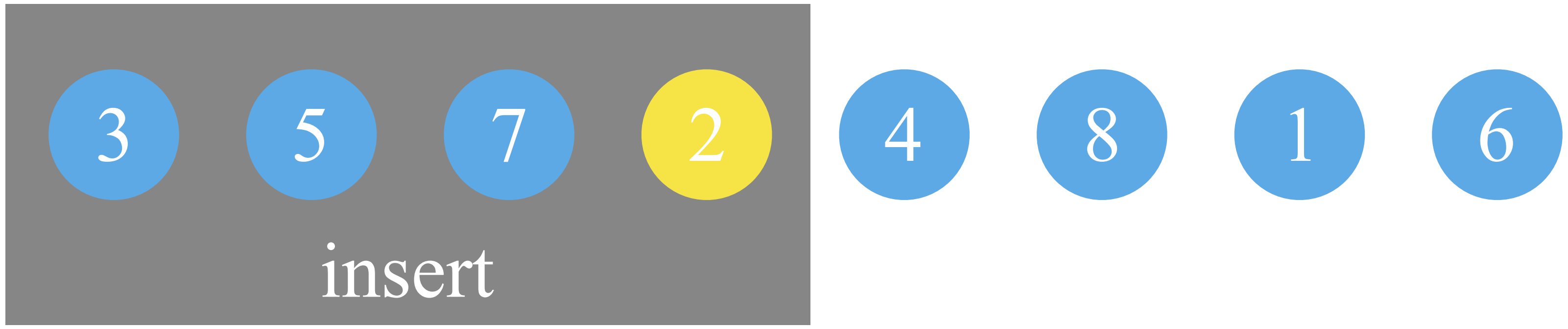
An instance

Insertion Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance

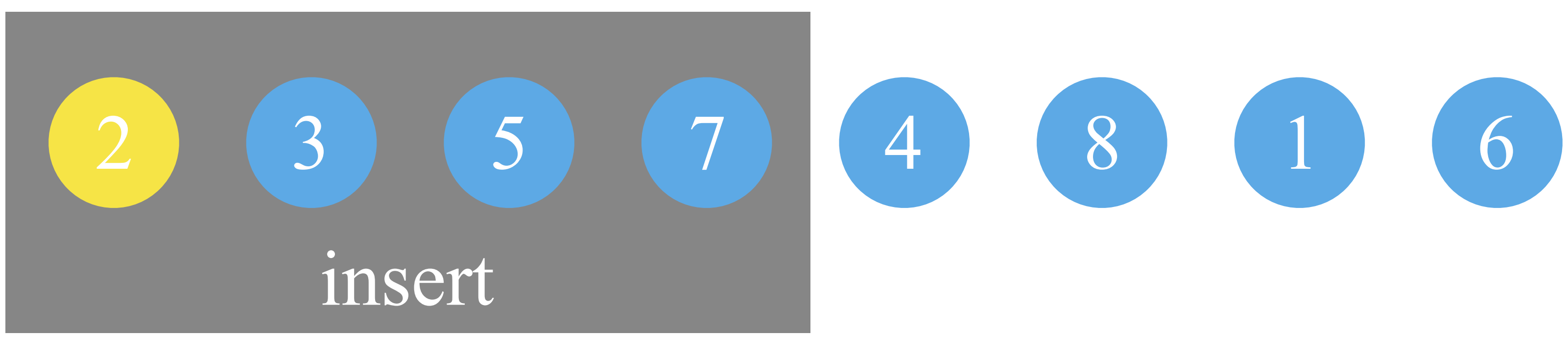


Insertion Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance



Insertion Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance

Insertion Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance

C++ Code

```
void insertion_sort(int *s, int n){

    for(int i=1; i<n; ++i){
        insert(s, i, s[i]);
    }
}
```

--- about the highlight ---

Again, we use a reduction here.

The running time is $O(n) \cdot O(n) = O(n^2)$. Why does this equality hold?

Exercises

Let $f(n) = O(n^a)$ and $g(n) = O(n^b)$ for some constants $a, b > 0$. Prove that

1. $f(n) + g(n) = O(n^c)$ for any $c \geq \max(a, b)$.

2. $f(n) \cdot g(n) = O(n^c)$ for any $c \geq a+b$.

Prove or disprove that

$$\sum_{i=1}^{n} f_i(n) = O(n) \text{ where } f_i = O(n) \text{ for every } i \in \{1, 2, \ldots, n\}.$$

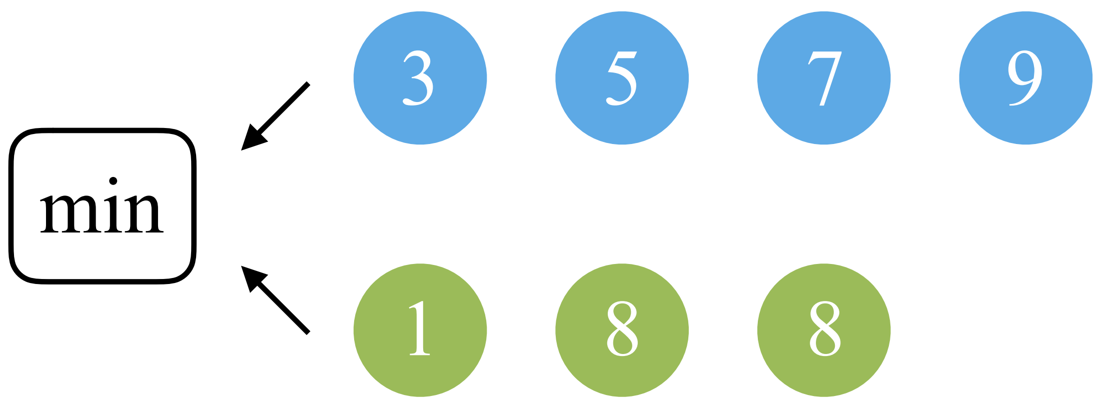
Merge Sort

Merge two sorted arrays into one

Input: two sorted arrays A and B of integers.

Output: a sorted array that comprises all elements in A and B.

An instance

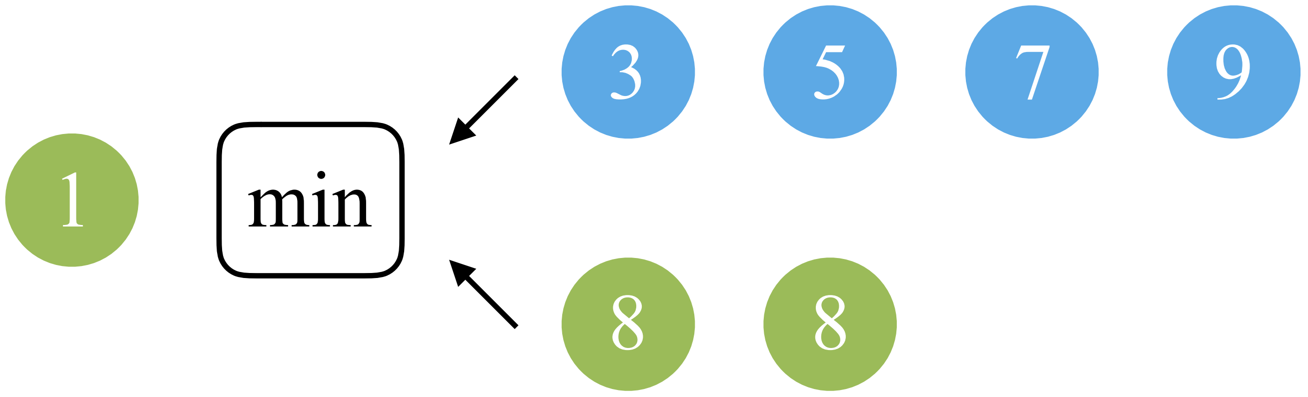


Merge two sorted arrays into one

Input: two sorted arrays A and B of integers.

Output: a sorted array that comprises all elements in A and B.

An instance

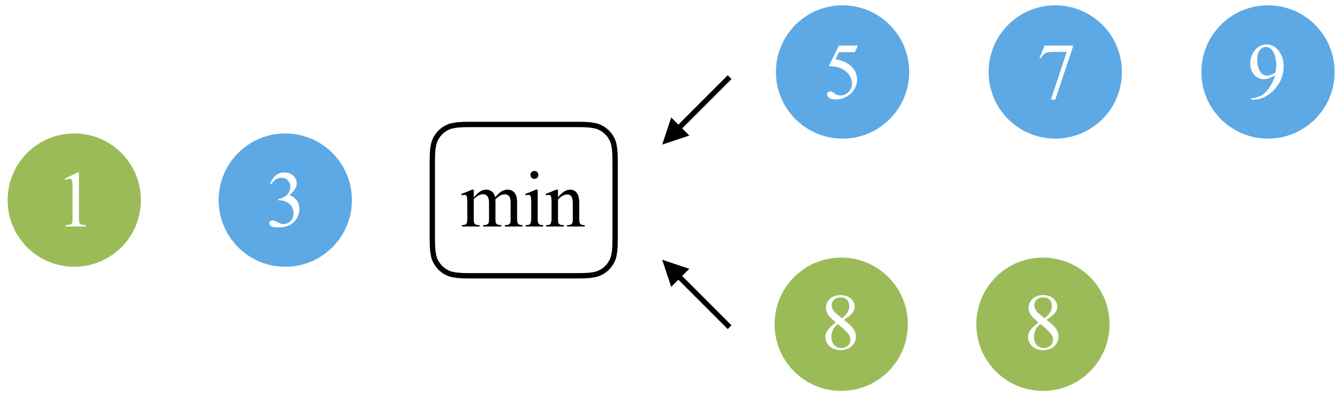


Merge two sorted arrays into one

Input: two sorted arrays A and B of integers.

Output: a sorted array that comprises all elements in A and B.

An instance

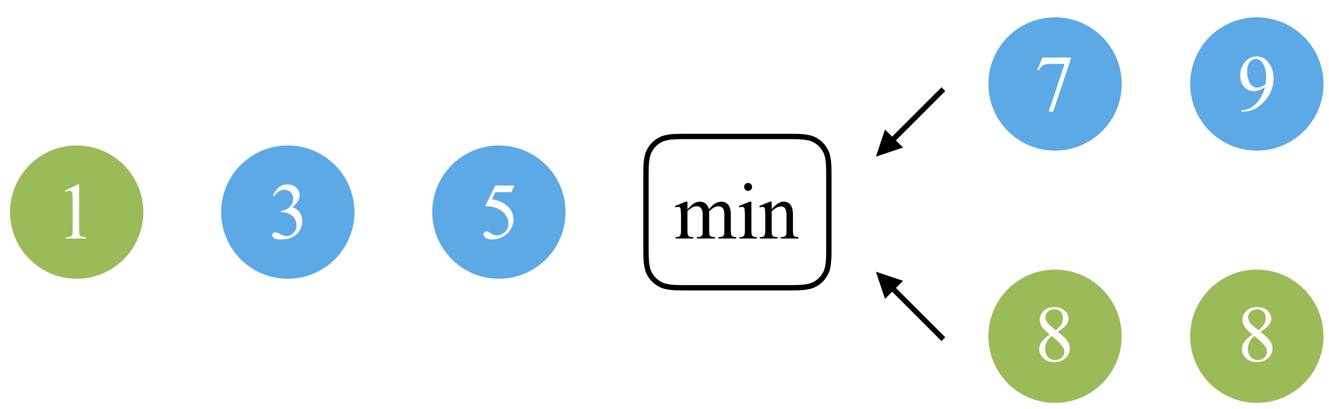


Merge two sorted arrays into one

Input: two sorted arrays A and B of integers.

Output: a sorted array that comprises all elements in A and B.

An instance



C++ Code

```
int* merge(int *s, int n, int *r, int m){

    int *ret = new int [n+m];
    int i = 0, j = 0, k = 0;

    while(1){
        if(i < n && j < m){ // when both arrays are not empty
            ret[k++] = ((s[i] < r[j]) ? s[i++] : r[j++]);
        }else{
            ret[k++] = ((i < n) ? s[i++] : r[j++]);
        }
    }
}
------
```

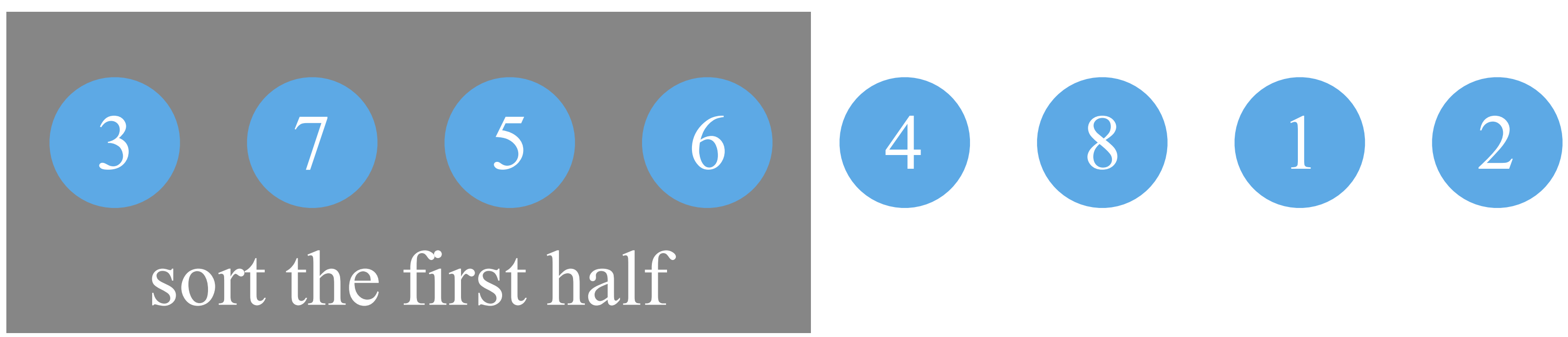
Merging two sorted arrays takes O(n+m) time.

Merge Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance

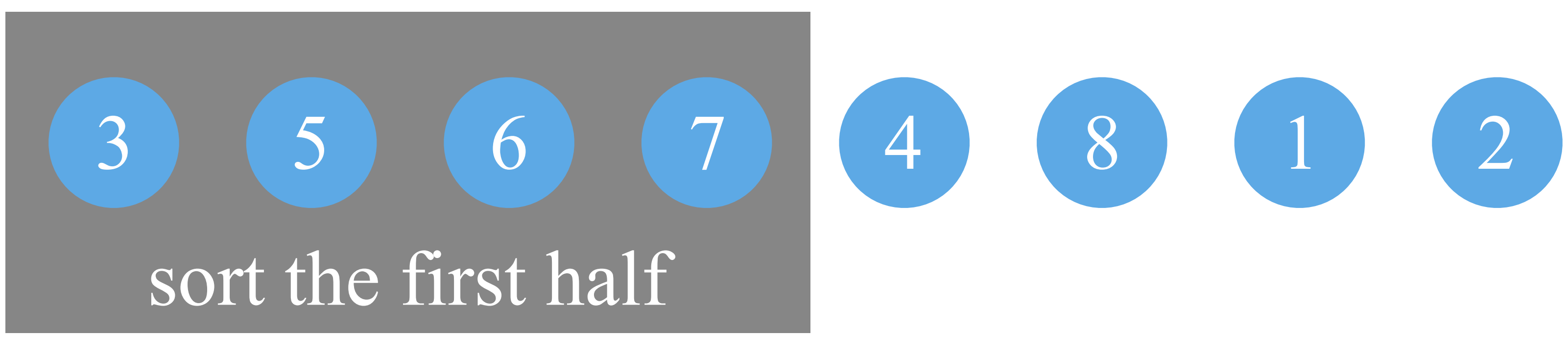


Merge Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance

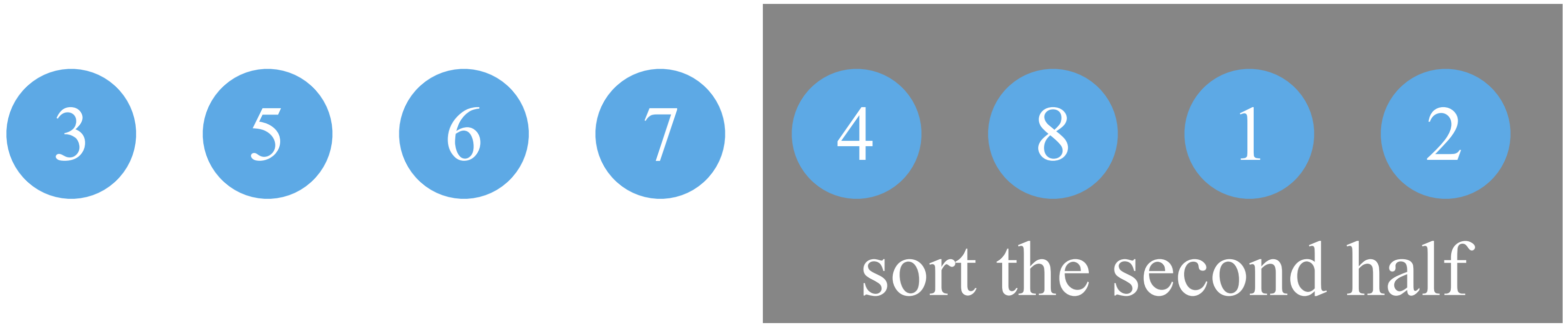


Merge Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance

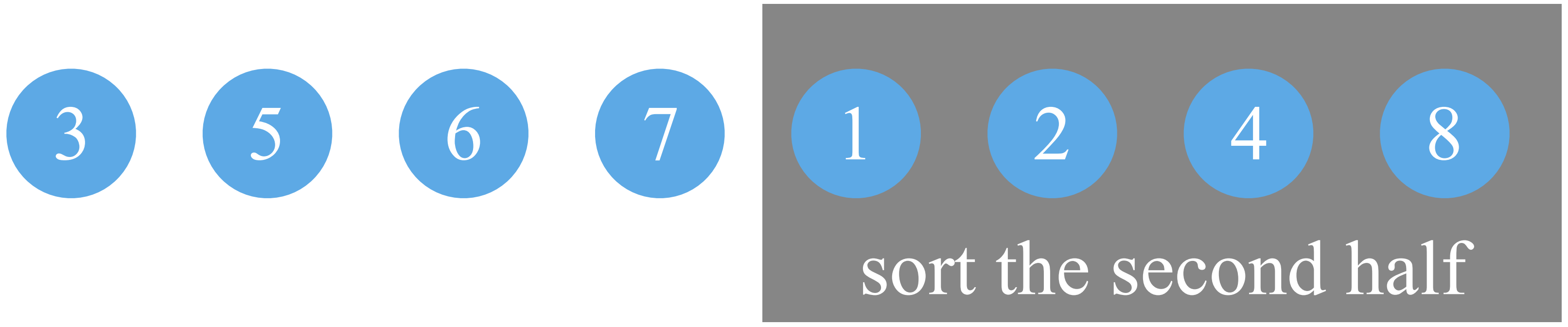


Merge Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance



Merge Sort

Input: an array A of n integers.

Output: the same array with the n integers ordered nondecrementally.

An instance

C++ Code

```
void merge_sort(int *s, int n){

    if(n == 1) return;

    int k = n/2;
    merge_sort(s, k);
    merge_sort(s+k, n-k);

    int *r = merge(s, k, s+k, n-k);
    memcpy(s, r, sizeof(int)*n);
}
```

--- about the highlight ---

A reduction from a problem to itself is called *recursion*. A recursion usually requires that the instance size decreases monotonically. Why?

C++ Code

```
void merge_sort(int *s, int n){

    if(n == 1) return;

    int k = n/2;
    merge_sort(s, k);
    merge_sort(s+k, n-k);

    int *r = merge(s, k, s+k, n-k);
    memcpy(s, r, sizeof(int)*n);
}
```

Merge sort needs at most $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_1 n$ operations where $T(1) = c_2$.

Substitution Method

Merge sort needs at most $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_1 n$ operations where $T(1) = c_2$. How to represent T(n) in terms of O(g(n))?

Guess $T(n) \leq d\, n \log n + d\, n$ for some constant $d > 0$.

// We will see how to guess.

For $n = 1$, $T(1) = c_2 \leq d \log 1 + d$ if we pick $d \geq c_2$.

Suppose $T(n) \leq d\, n \log n + d\, n$ for every $n < k$.

For $n = k$,

$$\begin{aligned} T(k) &= d(\lfloor k/2 \rfloor \log \lfloor k/2 \rfloor + \lceil k/2 \rceil \log \lceil k/2 \rceil) + c_1 k \\ &\leq d(\lfloor k/2 \rfloor + \lceil k/2 \rceil) \log k + c_1 k \\ &= dk \log k + c_1 k \\ &\leq dk \log k + dk \qquad \text{(if we pick } d \geq c_1\text{)} \end{aligned}$$

By induction on k, our guess is correct. Thus, $T(n) = O(n \log n)$.

Recursion-Tree Method

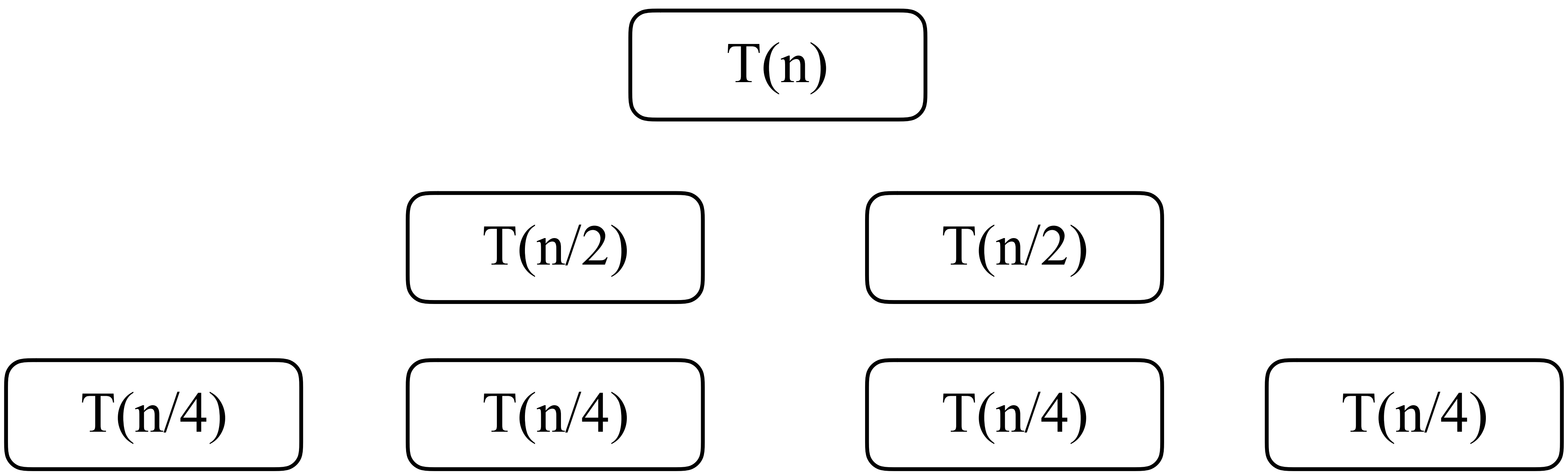
Merge sort needs at most $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + c_1 n$ operations where $T(1) = c_2$. We have seen how to verify the guess $T(n) = O(n \log n)$.
How to come up with a guess?

We simply need a guess, so we may drop the floor and the ceiling functions, and ignore the constants. We get:

$$T(n) = \begin{cases} 2T(n/2) + n & \text{if } n > 1 \\ 1 & \text{otherwise} \end{cases}$$

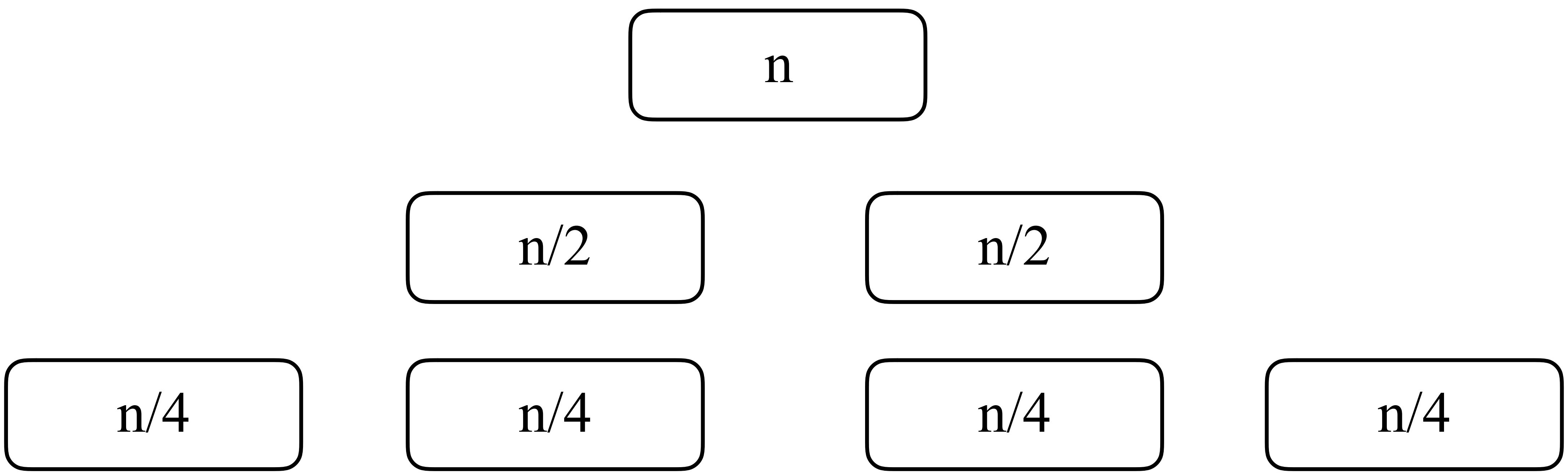
Recursion-Tree Method

$$T(n) = \begin{cases} 2T(n/2) + n & \text{if } n > 1 \\ 1 & \text{otherwise} \end{cases}$$



Recursion-Tree Method

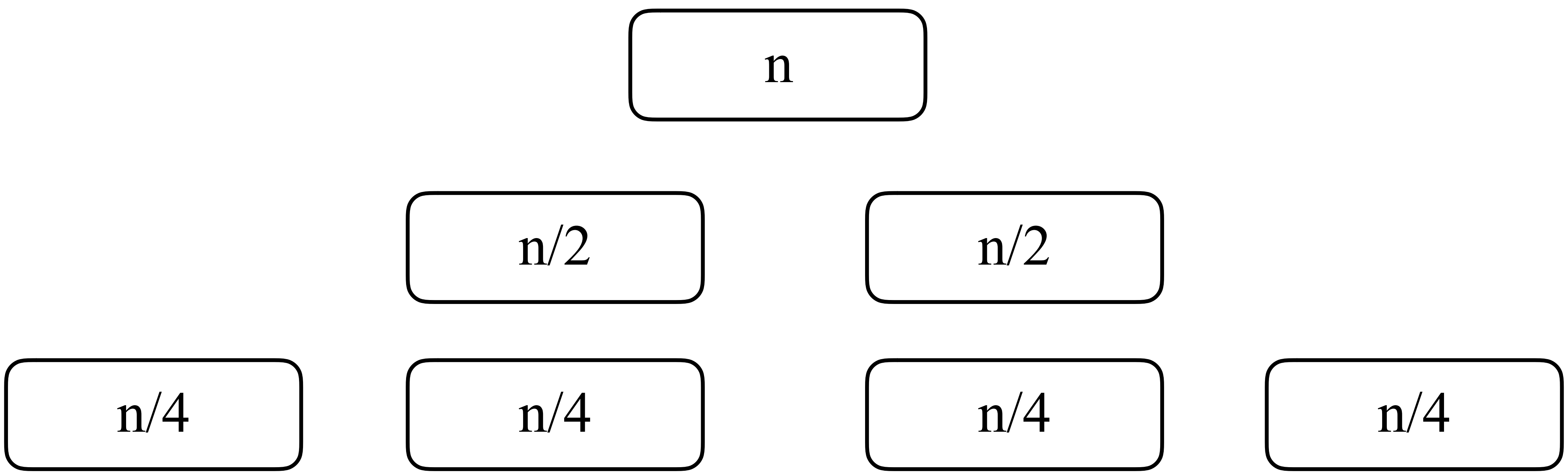
$$T(n) = \begin{cases} 2T(n/2) + n & \text{if } n > 1 \\ 1 & \text{otherwise} \end{cases}$$



Analyze the cost for each subproblem T(k)
without considering its recursive calls.

Recursion-Tree Method

$$T(n) = \begin{cases} 2T(n/2) + n & \text{if } n > 1 \\ 1 & \text{otherwise} \end{cases}$$



There are $\log_2 n$ layers, and for each layer the sum of cost is n. Consequently, the total cost is O(n log n).

Exercises

1. To merge k sorted length-n arrays into a single sorted one, can one do this in O(k n log k) time?

2. Tower of Honai is a classical example for recursion. An interesting variation can be found in Ex 4, Chap 1, JfA.