Recitation 2

Reminders:

Sorting Algorithms

INSERTION SORT

- Write a global function that takes an integer array and sort it in ascending order. It should traverse the array **from the last position to the first position**.

- Let's define the function declaration, at first.

```
void reversedInsertionSort (int *theArray, int n);
```

- Example:

23	78	45	8	32	56	Original List

Unsorted Part: 23 78 45 8 32 — Sorted Part: 56

23	78	45	8	32	56	Pass 1

Unsorted Part: 23 78 45 8 — Sorted Part: 32 56

23	78	45	8	32	56	Pass 2

Unsorted Part: 23 78 45 — Sorted Part: 8 32 56

23	78	8	32	45	56	Pass 3

Unsorted Part: 23 78 — Sorted Part: 8 32 45 56

23	8	32	45	56	78	Pass 4

Unsorted Part: 23 — Sorted Part: 8 32 45 56 78

8	23	32	45	56	78	Pass 5

Sorted Part: 8 23 32 45 56 78

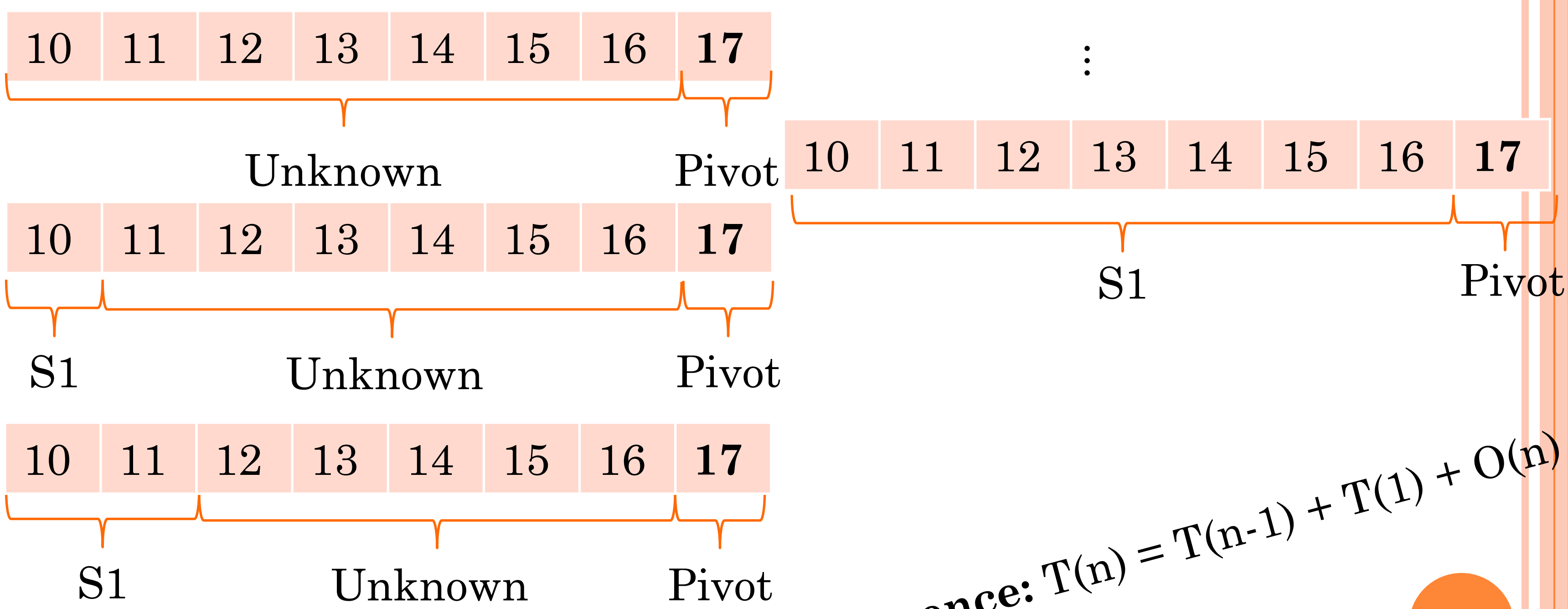
- What about its complexity?
  - Best Case: $O(n)$
    - Occurs when the array is already sorted.
  - Worst Case: $O(n^2)$
    - Occurs when the array is reversed sorted.
  - Average Case: $O(n^2)$
- What is the running time of the insertion sort if all keys are equal?
  - $O(n)$

```
void reversedInsertionSort (int *theArray, int n) {
	for (int i = n-2; i>=0; i--) {
		int nextItem = theArray[i];
		int j = i;
		while (j<n-1 && theArray[j+1]<nextItem) {
			theArray[j] = theArray[j+1];
			j++;
		}
		theArray[j] = nextItem;
	}
}
```

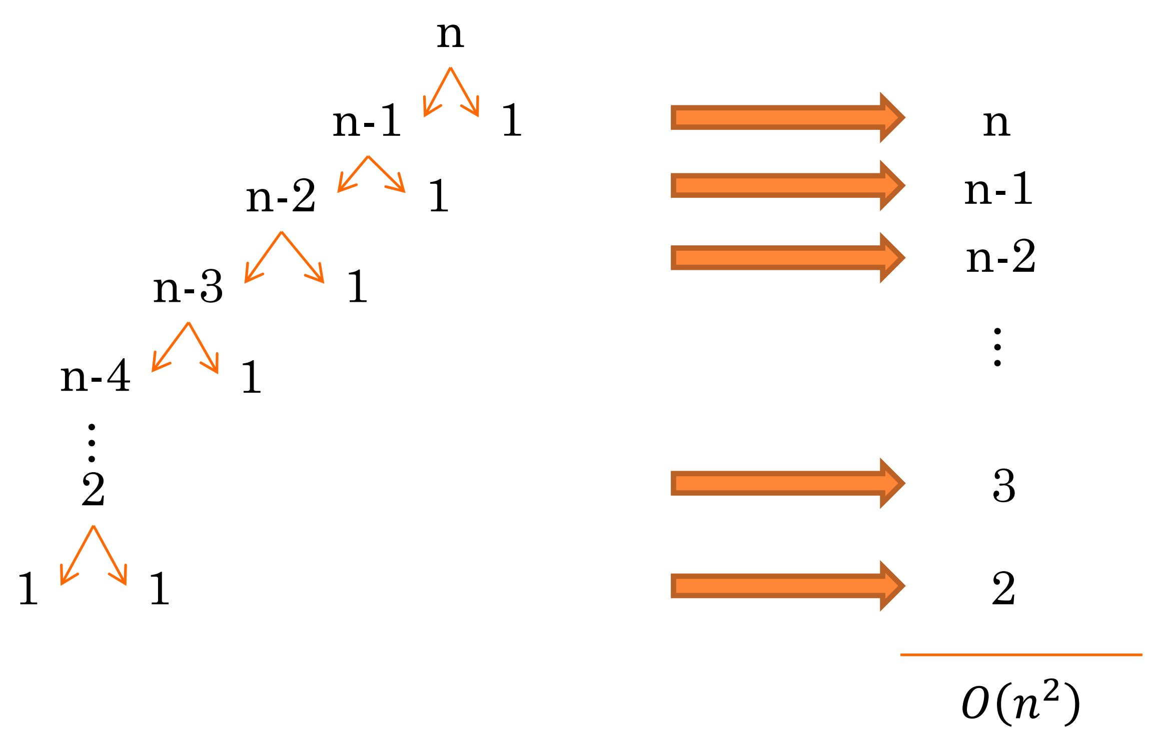
Quick Sort (Pivot Selection)

a. Sorted Input (ascending)

i. Pivot: The last element:



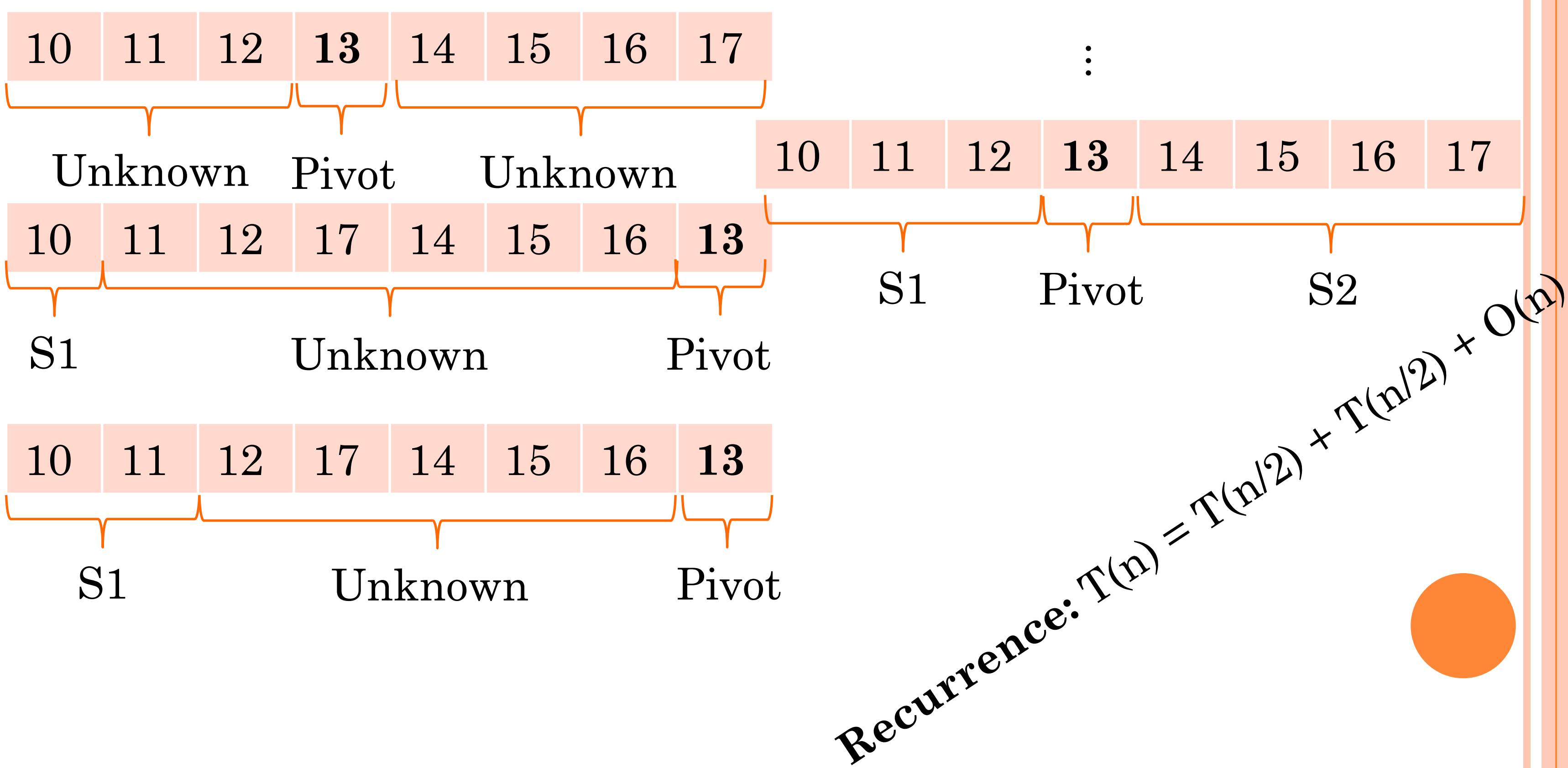
Recurrence: $T(n) = T(n-1) + T(1) + O(n)$

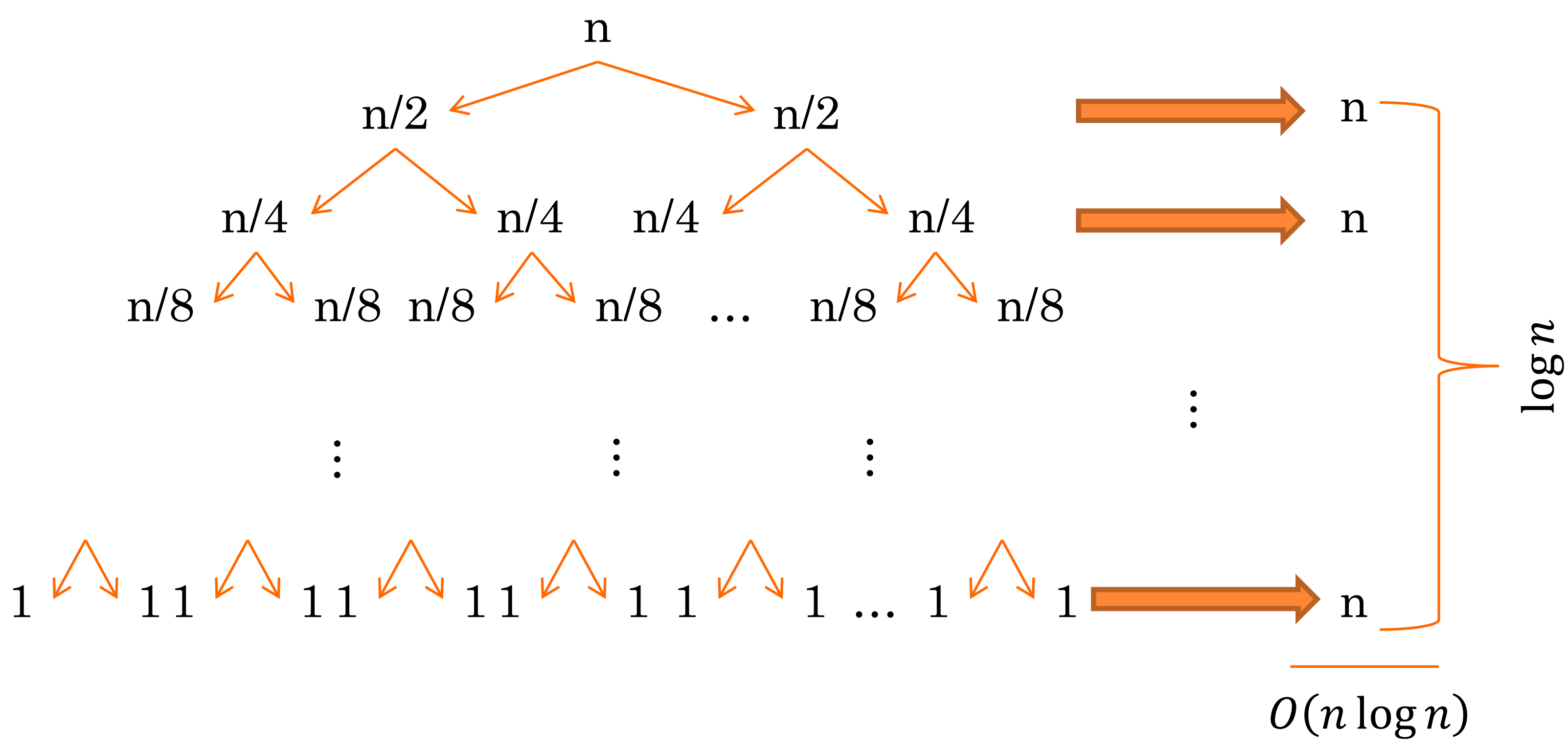

n
n-1
1
n-2
1
n-3
1
n-4
1
⋮
2
1
1
n
n-1
n-2
⋮
3
2
$O(n^2)$

Quick Sort (Pivot Selection)

a. Sorted Input (ascending)

ii. Pivot: the average of all keys

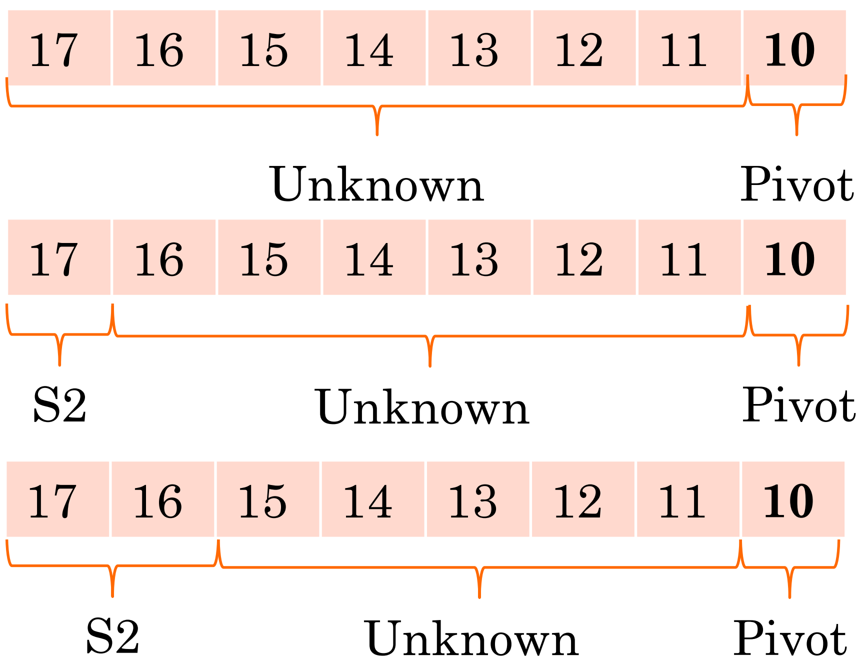


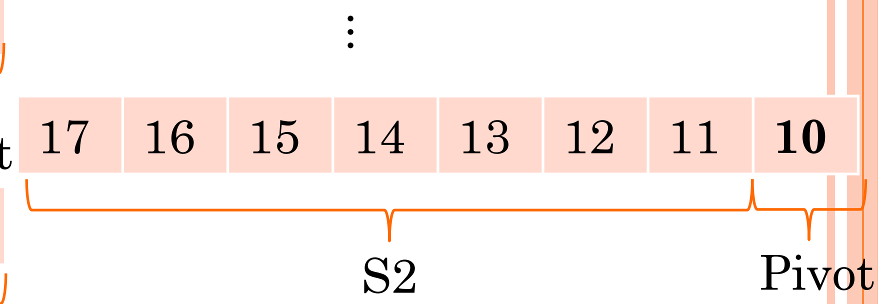

n
n/2
n/2
n/4
n/4
n/4
n/4
n/8
n/8
n/8
n/8
…
n/8
n/8
1
1 1
1 1
1 1
1 1
1 … 1
1
n
n
n
log n
O(n log n)

Quick Sort (Pivot Selection)

b. Sorted Input (descending)

i. <u>Pivot: The last element:</u>





Recurrence: T(n) = T(1) + T(n-1) + O(n)

Complexity: $O(n^2)$

Quick Sort (Pivot Selection)

b. Sorted Input (descending)

   ii. <u>Pivot: the average of all keys</u>

17	16	15	**14**	13	12	11	10

Unknown (17, 16, 15) — Pivot (14) — Unknown (13, 12, 11, 10)

17	16	15	10	13	12	11	**14**

S2 (17) — Unknown (16, 15, 10, 13, 12, 11) — Pivot (14)

17	16	15	10	13	12	11	**14**

S2 (17, 16) — Unknown (15, 10, 13, 12, 11) — Pivot (14)

⋮

17	16	15	**14**	13	12	11	10

S2 (17, 16, 15) — Pivot (14) — S1 (13, 12, 11, 10)

Recurrence: $T(n) = T(n/2) + T(n/2) + O(n)$

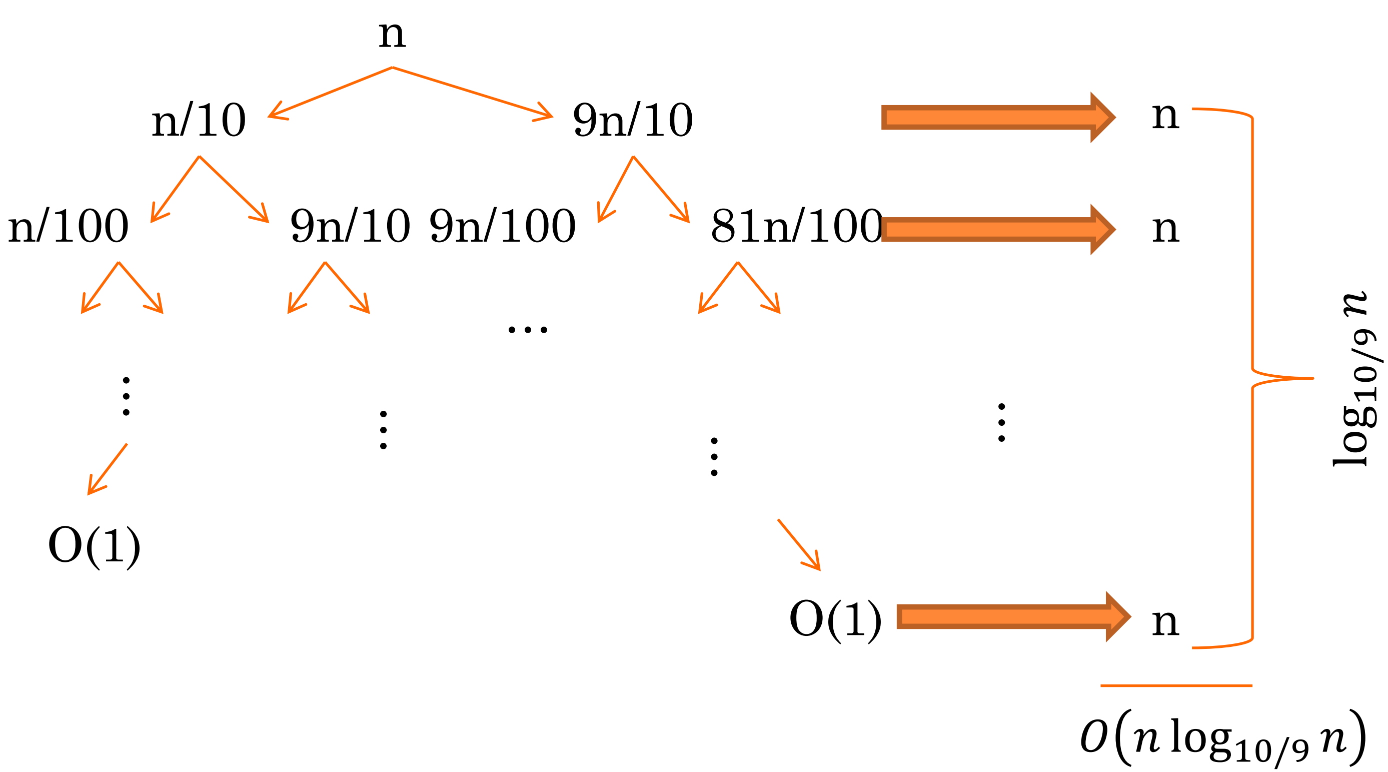
Complexity: $O(n \log n)$

Quick Sort (Pivot Selection)

c. Random Input

- Choosing pivot; the first element, the last element or a random key does **not** matter.
- What if the split is always $^{1}/_{10}$: $^{9}/_{10}$?

Recurrence: $T(n) = T(n/10) + T(9n/10) + O(n)$



$$n \log_{10} n \leq \mathrm{T(n)} \leq n \log_{10/9} n \Rightarrow \boldsymbol{O(n \log n)}$$

- Average case analysis:

- T(n) = 1/n (T(1) + T(n-1))
  + 1/n (T(2) + T(n-2))
  + 1/n (T(3) + T(n-3))
  …
  …
  + 1/n (T(n-1) + T(1))
  + O(n)

$$T(n) = \frac{1}{n}\sum_{k=1}^{n-1}\big(T(k) + T(n-k)\big) + O(n)$$

$$T(n) = \frac{2}{n}\sum_{k=1}^{n-1} T(k) + O(n)$$

solving by substitution method

$$\mathbf{T(n) = O(n\log n)}$$

Merge Sort

- Suppose that you remove the call to merge from mergesort algorithm obtain:

```
mystery (inout theArray:ItemArray, in n: integer) {
        //mystery algorithm for theArray[0…n-1]
        if(n>1) {
                mystery(lefthalf(theArray), n/2)
                mystery(righthalf(theArray), n/2)
        }
}
```

- What does this algorithm do?

Radix Sort

```
radixSort(inout theArray:ItemArray, in n:integer, in d:integer)
{
//sort n d-digit integers in the array theArray
          for (j=d down to 1) {
                    Initialize 10 groups to empty
                    Initialize a counter for each group to 0
                    for (i=0 through n-1) {
                              k=j th digit of theArray[i]
                              Place theArray[i] at the end of group k
                              Increase k th counter by 1
                    }
          Replace the items in theArray with all items in group;
0,…,8,9
}
```

Radix Sort (Cont'd)

- How many groups do we need for binary and hexadecimal radix sort?
  - 2 for binary (each digit can be either 0 or 1)
  - 16 for hexadecimal (each digit can be one of the symbols in set {[0-9] U {A,B,C,D,E,F}}
- What is the suitable data structure for radix sort?
  - Hash table

Exercise

- Question : Write a method to print key values of all pairs of given array satisfying following condition.
- Condition: The sum of two integers in a pair is equal to given key.
- Example:

Array:

12	7	6	3	9	5	8	2	5

Key:

15

Output:

(12,3)	(7,8)	(6,9)

Naive approach: For each element, scan the whole array. $\boldsymbol{O(n^2)}$

- Efficient solution:

```
theMethod (in theArray:IntegerArray, in n:integer, in key:integer) {
        //sort theArray using any O(nlogn)-sorting algorithm.
        MergeSort (theArray, n);
        for (currentIndex = 0; currentIndex < n; currentIndex++) {
                currentKey = theArray[currentIndex];
                searchKey = key - currentKey;
                newIndex  = BinarySearch (theArray, n, searchKey);
                if (newIndex != -1)
                        print ( currentKey, searchKey);
        }
}
```

$O(n \log n)$ (MergeSort)

$O(n \log n)$ (for loop)

$$T(n) = O(n \log n)$$