L14: Sorting

Alex Steiger

CompSci 201: Spring 2024

2/28/2024: LEAP DAY EVE

1

Announcements, Coming up

- Today, Wednesday 2/28
  - APT 5 (linked list problems) due
- Next Monday 3/4
  - Project P3: DNA (linked list project) due
- Next Wednesday 3/6
  - APT 6 (sorting problems) due
- Then…Spring Break!

2

Today's outline

1. Announce Midsemester Survey
   1. Invaluable for staff, especially UTAs ⇒ for you!
   2. Look for Canvas announcement from Violet
2. Sorting in Java: Comparing objects with `Comparable` and `Comparator`
3. Efficient sorting algorithms
   1. Insertion sort
   2. Recursive Mergesort

3

Sorting in Java: Comparable, Comparator

4

Sorting w/ Java.util: Put elements of Array/List in non-decreasing order

- **Arrays.sort** / **Collections.sort** are void – they sort the array/list passed as an argument.
- Default order is non-decreasing (least to greatest).

```
67      int[] elements = {5, 3, 9, 2, 4, 1};
68      Arrays.sort(elements);
69      System.out.println(Arrays.toString(elements));
```

- Prints **[1, 2, 3, 4, 5, 9]**

5

Java API Sort Algorithms

- `Collections.sort` (for a List)
- `Arrays.sort` (for an Array)

- Both O(N log(N)), ***nearly linear*** runtime complexity.
- Sorts in-place, mutates the input rather than return a new List/Array.
- ***Stable***, does not reorder elements if not needed (e.g., if two elements are equal).

6

What can be compared and sorted in Java?

- Objects of a Class that implements [`Comparable` interface](). Has a **`naturalOrder`**.
- Requires implementing a **`.compareTo()`** method

Should return an int:

- < 0 if **`this`** comes before the parameter.
- 0 if **`this`** and the parameter are equal.
- > 0 if **`this`** comes after the parameter.

```
private static class Person implements Comparable<Person> {
    String first;
    String last;
    public Person(String s) {...}
    public String getLast() { return last; }
    public String getFirst() { return first; }
    public String toString() { return first + " " + last; }
    @Override
    public int compareTo(Person p){
        int diff = last.compareTo(p.last);
        if (diff != 0) return diff;
        return first.compareTo(p.first);
    }
}
```


7

Strings are Comparable

- What is the equivalent of < for Strings?
- Use the **`compareTo`** method for the natural lexicographic (dictionary/sorted) ordering.

```
jshell> "a".compareTo("b");
$30 ==> -1
```
Negative for "less than"

```
jshell> "b".compareTo("b");
$31 ==> 0
```
Zero for "equal"

```
jshell> "b".compareTo("a");
$32 ==> 1
```
Positive for "less than"

```
jshell> "az".compareTo("cb");
$37 ==> -2
```
Lexicographic, check first character, second if equal, third if still equal, …

8

Sorting `Comparable` objects by `naturalOrder`

`[sloth, house, owl, ant, mice, kelp]`

```
String[] a = {"sloth", "house", "owl", "ant", "mice", "kelp"};
System.out.println(Arrays.toString(a));

String[] copy = Arrays.copyOf(a, a.length);
Arrays.sort(copy);
System.out.println(Arrays.toString(copy));
```

`[ant, house, kelp, mice, owl, sloth]`

- `naturalOrder` for Strings is lexicographic (alphabetical or dictionary order)

9

Comparable for other classes?

All Blob comparing code available here

- Can implement `Comparable` interface when defining your own class.

```
3   public class Blob implements Comparable<Blob> {
4       String name;
5       String color;
6       int size;
```

- Must implement a `compareTo` method

```
14      @Override
15      public int compareTo(Blob other) {
16          return this.name.compareTo(other.name);
17      }
```

Compares blobs by their names

10

Sorting Comparable Objects

- Running code in a main method…

```
40      System.out.println(myBlobs);
```

Original: [(bo, blue, 4), (al, red, 2), (cj, green, 1), (di, red, 4)]

```
42      Collections.sort(myBlobs);
43      System.out.println(myBlobs);
```

Sorted: [(al, red, 2), (bo, blue, 4), (cj, green, 1), (di, red, 4)]

- Formal guarantee: Element `e1` will come before `e2` (after sorting) if `e1.compareTo(e2) < 0`.

11

Defining a Comparator

- What if…
  - The class doesn't implement Comparable?
  - Or you want to sort a different way?
- Create a helper class that implements the `Comparator` interface.
  - One method: `compare`: indicates how to compare two objects
- Then pass a Comparator object to your call to sort.

12

Defining a `Comparator<Blob>`

```
1   import java.util.Comparator;
```

Separate class:
- implements Comparator<TypeToCompare>,
- and implements a single method compare

```
 8  public class BlobComparator implements Comparator<Blob> {
 9      @Override
10      public int compare(Blob a, Blob b) {
11          int sizeDiff = a.size - b.size;
12          if (sizeDiff != 0) {
13              return (-1) * sizeDiff;
14          }
15          return a.compareTo(b);
16      }
17  }
```

Takes 2 parameters, Should return:
- < 0 if a comes before b,
- > 0 if a comes after b,
- 0 if equal in order

Flipping the sign reverses the comparison, large to small

Breaking ties by the natural order

13

Sorting with a Comparator

- Running code in a main method…

```
40      System.out.println(myBlobs);
```

Original: [(bo, blue, 4), (al, red, 2), (cj, green, 1), (di, red, 4)]

Create a `BlobComparator` object, pass it to the sort.

```
48      Collections.sort(myBlobs, new BlobComparator());
49      System.out.printf(format: "%s\n\n", myBlobs);
```

Sorted: [(bo, blue, 4), (di, red, 4), (al, red, 2), (cj, green, 1)]

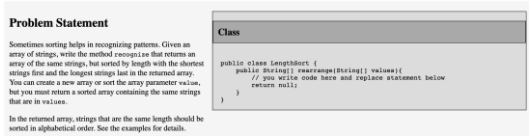
- Element `e1` will come before `e2` (after sorting) if `compare(e1, e2) < 0`.

14

Private Inner Comparator

- Can define a Comparator class as a private inner class if only used inside the class.
- Useful for APTs, here is an example:

SimpleSort APT



Problem Statement

Sometimes sorting helps in recognizing patterns. Given an array of strings, write the method `recognize` that returns an array of the same strings, but sorted by length with the shortest strings first and the longest strings last in the returned array. You can create a new array or sort the array parameter `value`, but you must return a sorted array containing the same strings that are in `values`.

In the returned array, strings that are the same length should be sorted in alphabetical order. See the examples for details.

Class

```
public class LengthSort {
    public String[] rearrange(String[] values){
        // you write code here and replace statement below
        return null;
    }
}
```


15

Private Inner Comparator

- Can define a Comparator class as a private inner class if only used inside the class.
- Useful for APTs, here is an example:

- Given `String[] values`:
  - Sort first in ***non-decreasing order of length***,
  - then sort same-length in ***alphabetical order***.

- ["a", "b", "c", "an", "be", "pi", "test", "quiz"]

16

Template for Solving LengthSort with a Private Inner Comparator

Can see this code here

```
1  import java.util.Arrays;
2  import java.util.Comparator;
3
4  public class LengthSort {
5      private class LengthSortComp implements Comparator<String> {
6          @Override
7          public int compare(String a, String b) {
8              // Need to modify this to solve the problem
9              return a.compareTo(b);
10         }
11     }
12
13     public String[] rearrange(String[] values){
14         Arrays.sort(values, new LengthSortComp());
15         return values;
16     }
17 }
```


17

`Comparable` vs. `Comparator`

- `Comparable a`: use `a.compareTo(b)`
  - What is method signature? One parameter
  - Method in class of which object **a** is an instance
  - **a** is **this**, **b** is a parameter
- `Comparator c`, use `c.compare(a,b)`
  - Method has two parameters
  - Part of `Comparator` (Java API link)
- Both return an int:
  - < 0 (means **a** comes before **b**)
  - == 0 (means **a** equals **b**)
  - > 0 (means **a** comes after **b**)

18

Runtime Complexity of Sort and Comparator?

- `Arrays.sort`, `Collections.sort`, call either `compareTo` (default) or `compare` (if you give a `Comparator`)…
- O(N log(N)) `compareTo/compares`, on an Array/List of N elements.
- Exists theoretical proof that this many comparisons is ***necessary*** for any comparison-based sorting.

19

When is comparing once not constant time?

```
4  public class ListComp implements Comparator<List<Integer>> {
5      @Override
6      public int compare(List<Integer> list1, List<Integer> list2) {
7          int minLength = Math.min(list1.size(), list2.size());
8          for (int i=0; i<minLength; i++) {
9              int diff = list1.get(i) - list2.get(i);
10             if (diff != 0) {
11                 return diff;
12             }
13         }
14         return 0;
15     }
16 }
```

Runtime complexity of this Comparator may depend on the length of the two Lists being compared.

Overall runtime complexity to sort N `ArrayList`s, each with M elements, is O(MN log(N)) in the worst case with this `Comparator`.

20

java.util.Comparator: Convenient Shorthands

- `Comparator.naturalOrder` and `reversed()`

```
jshell> Comparator<String> c = Comparator.naturalOrder()
c ==> INSTANCE

jshell> c.compare("a","b")
$12 ==> -1

jshell> c.reversed().compare("a","b")
$13 ==> 1
```

Must be `Comparable`

- `Comparator.comparing`

```
jshell> Comparator<String> c = Comparator.comparing(String::length)
c ==> java.util.Comparator$$Lambda$27/0x0000000800b97c00@2b71fc7e

jshell> c.compare("this", "is")
$15 ==> 1

jshell> c.compare("is", "it")
$16 ==> 0
```

Syntax is: `<Type>::<method name>` to sort something of the Type by the result of some getter method that returns something Comparable.

21

`Comparator`-generating shorthands

`[sloth, house, owl, ant, mice, kelp]`

```
copy = Arrays.copyOf(a, a.length);
Arrays.sort(copy, Comparator.comparing(String::length));
System.out.println(Arrays.toString(copy));
```

`[owl, ant, mice, kelp, sloth, house]`

- Why does "owl" come before "ant"?
  - Stable sort respects order of equal keys

22

Using `.thenComparing` shorthand

`[sloth, house, owl, ant, mice, kelp]`

```
Arrays.sort(copy, Comparator.
                    comparing(String::length).
                      thenComparing(Comparator.naturalOrder()));
```

`[ant, owl, kelp, mice, house, sloth]`

- First compare by length
  - if same? Compare naturally

23

Comparator with "lambdas"

- Can also define a comparator with a *"lambda expression."*

```
Integer[] nums = {2, 0, 1};
Comparator<Integer> comp = (a, b) -> (b-a);
```

Type we want to compare

Given an `a` and a `b` of that type…

`comp.compare(a,b)` should return this expression

```
Arrays.sort(nums, comp);
```

nums is now { 2, 1, 0 }

24

What is printed by the following line of code?

```
System.out.println("duke".compareTo("devils"));
```

- true
- false
- an integer less than 0
- 0
✓- an integer greater than 0

26

After sorting, ar will be...

```
String[] ar = {"bird", "dog", "cat", "snake"};
Comparator<String> comp = Comparator.comparing(String::length);
Arrays.sort(ar, comp);
```

Ans: [dog, cat, bird, snake]

27

Suppose you have the following list of lists of integers:

[[2, 0, 1], [1, 0, 1], [1, 6]]. After sorting, the list would be ordered as...

```
4  public class ListComp implements Comparator<List<Integer>> {
5      @Override
6      public int compare(List<Integer> list1, List<Integer> list2) {
7          int minLength = Math.min(list1.size(), list2.size());
8          for (int i=0; i<minLength; i++) {
9              int diff = list1.get(i) - list2.get(i);
10             if (diff != 0) {
11                 return diff;
12             }
13         }
14         return 0;
15     }
16 }
```

Ans: [[1, 0, 1], [1, 6], [2, 0, 1]]

28

Suppose you have an **ArrayList** myLists of N **ArrayLists**, each of size at most M. The worst-case runtime complexity to compare any two elements of myLists would be….

```
4  public class ListComp implements Comparator<List<Integer>> {
5      @Override
6      public int compare(List<Integer> list1, List<Integer> list2) {
7          int minLength = Math.min(list1.size(), list2.size());
8          for (int i=0; i<minLength; i++) {
9              int diff = list1.get(i) - list2.get(i);
10             if (diff != 0) {
11                 return diff;
12             }
13         }
14         return 0;
15     }
16 }
```

Ans: O(M)

29

Given an Array of N Strings, each of length at most M, the worst case runtime complexity to sort the Array with java.util.Arrays.sort is..

Ans: O(M N log N)

30

Efficient sorting algorithms

See example implementations here

31

Selection Sort with a Loop Invariant

- Loop invariant: On iteration `i`, the first `i` elements are the smallest `i` elements in sorted order.
- On iteration `i`…
  - Find the smallest element from index `i` onward
    - *(By loop invariant, must be the **next smallest** element)*
  - Swap that with the element at index `i`
- Algorithm is called ***Selection Sort***.

8
5
2
6
9
3
1
4
0
7

32

Selection Sort Code and Runtime

```
3   public static void selectSort(int[] ar) {
4       for (int i=0; i<ar.length; i++) {
5           int minDex = i;
6           for (int j=i+1; j<ar.length; j++) {
7               if (ar[j] < ar[minDex]) {
8                   minDex = j;
9               }
10          }
11          int temp = ar[i];
12          ar[i] = ar[minDex];
13          ar[minDex] = temp;
14      }
15  }
```

Nested O(N) loops, overall $O(N^2)$

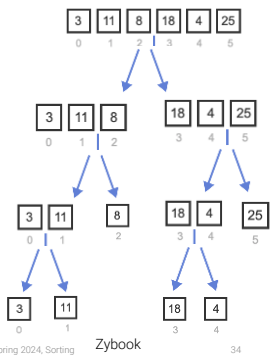
8
5
2
6
9
3
1
4
0
7

33

Mergesort

High level idea:

- Base case: size 1
  - Return list
- Recursive case:
  - Mergesort(first half)
  - Mergesort(second half)
  - …



Zybook

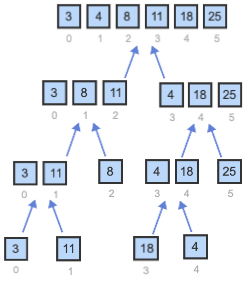
34

Mergesort

High level idea:

- Base case: size 1
  - Return list
- Recursive case:
  - Mergesort(first half)
  - Mergesort(second half)
  - Merge the sorted halves
  - Return sorted

Helper method



Zybook

35

Mergesort recursive wrapper

- A recursive wrapper method:
  - Is the top-level method a user would call,
  - Is not itself recursive, but makes the initial call to a recursive method,
  - Allows recursive helper method to have additional parameters.

```
30  public static void mergeSort(int[] ar) {
31      mergeHelper(ar, l: 0, ar.length);
32  }
```

Want to specify a left and right boundary of the subarray for each recursive call to sort

36

Mergesort recursive method

- Should sort everything in **ar** starting at index **l** and up to (but not including) index **r**.

```
34  public static void mergeHelper(int[] ar, int l, int r) {
35      int diff = r-l;
36      if (diff < 2) { return; }
37      int mid = l + diff/2;
38      mergeHelper(ar, l, mid);
39      mergeHelper(ar, mid, r);
40      merge(ar, l, mid, r);
41  }
```

Base case, if 0 or 1 elements, nothing to do

Recursively sort 1st half

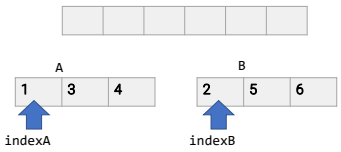
Recursively sort 2nd half

Merge the 2 sorted parts

37

Merge method concept

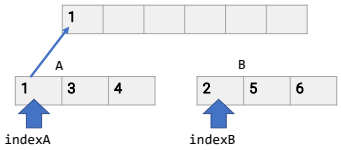
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



38

Merge method

- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



39

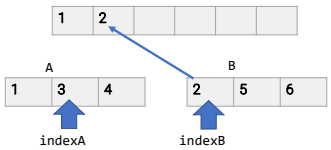
Merge method

- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.

40

Merge method

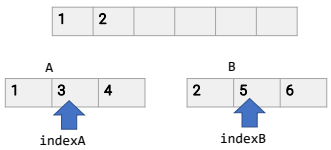
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



41

Merge method

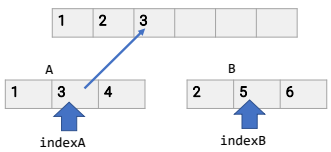
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



42

Merge method

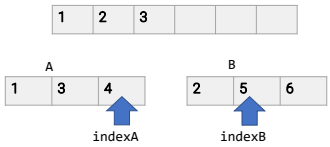
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



43

Merge method

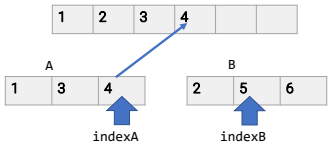
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



44

Merge method

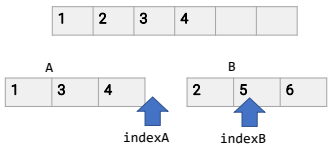
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



45

Merge method

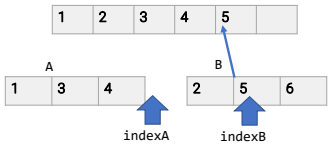
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



46

Merge method

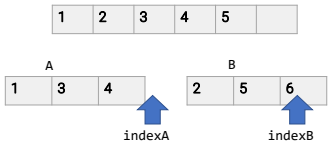
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



47

Merge method

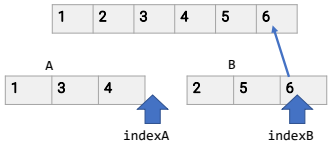
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



48

Merge method

- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



49

Merge method initialization

- Should merge **ar[l…mid]** and **ar[mid…r]**

```
43  public static void merge(int[] ar, int l, int mid, int r) {
44      int[] sorted = new int[r-l];
45      int sDex=0; int lDex=l; int rDex=mid;
```

- Need a new array **sorted** to put the merged results in, will copy back over **ar** later.
- Keeping track of 3 indices:
  - **sDex** = where we are in the **sorted** array
  - **lDex** = where we are in **ar[l…mid]**
  - **rDex** = where we are in **ar[mid…r]**

50

Merge method loop

```
46  while (lDex < mid && rDex < r) {
47      if (ar[lDex] <= ar[rDex]) {
48          sorted[sDex] = ar[lDex];
49          lDex++;
50      }
51      else {
52          sorted[sDex] = ar[rDex];
53          rDex++;
54      }
55      sDex++;
56  }
```

While something left in ar[l…mid] and ar[mid…r]

Add the smaller element and increment its index.

Increment **sDex** in either case

51

Finishing the merge method

- Will finish with **ar[l…mid]** or **ar[mid…r]** first, need to copy the rest of the other.
- Then need to copy sorted back onto **ar[l…r]**

```
57  if (lDex == mid) { System.arraycopy(ar, rDex, sorted, sDex, r-rDex); }
58  else { System.arraycopy(ar, lDex, sorted, sDex, mid-lDex); }
59  System.arraycopy(sorted, srcPos: 0, ar, l, r-l);
```

- Code uses the System.arraycopy method:

```
public static void arraycopy(Object src,
                             int srcPos,
                             Object dest,
                             int destPos,
                             int length)
```

Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array. A subsequence of

52

Is this any faster? Empirically…

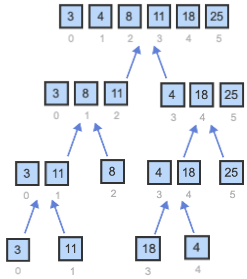
N (thousands)	Selection sort (ms)	Insertion sort(ms)	Merge sort (ms)	Java.util Arrays.sort (ms)
10k	22	40	1	2
30k	168	334	2	2
90k	1481	967	7	6
270k	13175	8716	22	14

Looks linear but not quite: O(N log(N)) is *nearly linear*.

53

Why mergesort is O(Nlog(N)), intuition

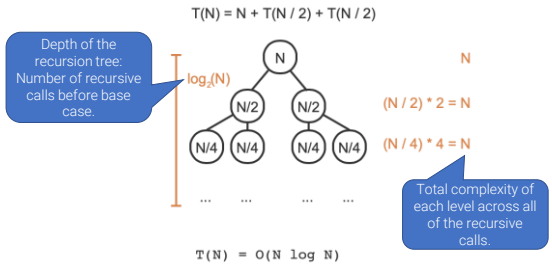
- Halves at each level, so just O(log(N)) levels.
- If we can do all of the merges at each level in O(N) time?
- Overall O(Nlog(N)).



Zybook

54

Recursion tree



Visualization from the Zybook

55

Recurrence Relations

56

Analyzing Recursive Runtime

Develop a recurrence relation of the form

$$T(N) = a \cdot T(g(N)) + f(N)$$

Total runtime | Recursive call(s) | Non-recursive runtime

- $T(N)$ - runtime of method with input size *N*
- a is the number of recursive calls
- $g(N)$ - how much input size decreases on each recursive call
- $f(N)$ - runtime of non-recursive code on input size N

57

Analyzing Runtime of Recursive Reverse

```
3   public static ListNode reverse(ListNode list) {
4       if (list == null || list.next == null) {
5           return list;
6       }
7       ListNode reversedLast = list.next;
8       ListNode reversedFirst = reverse(list.next);
9       reversedLast.next = list;
10      list.next = null;
11      return reversedFirst;
12  }
```

$g(N) = N - 1$

$f(N) = O(1)$

$$T(N) = T(N-1) + O(1)$$

58

Solving Recurrence Relations

$$T(N) = T(N-1) + 1$$
$$= (T(N-2) + 1) + 1$$
$$= T(N-3) + 3$$
$$\vdots$$
$$= T(1) + N$$
$$= O(N)$$

Apply recurrence again to T(N-1)

Total runtime

And again, to T(N-2)

T(1) is base case, just O(1)

59

Recurrence relations and expectations in 201

- In general, will **not** be asked to solve recurrence relations on exams (for later classes in theory).
- You **will** be asked to determine the recurrence relation of a given algorithm/code.

Recurrence	Algorithm	Solution
`T(n) = T(n/2) + O(1)`	binary search	`O(log n)`
`T(n) = T(n-1) + O(1)`	sequential search	`O(n)`
`T(n) = 2T(n/2) + O(1)`	tree traversal	`O(n)`
`T(n) = T(n/2) + O(n)`	qsort partition ,find k^{th}	`O(n)`
`T(n) = 2T(n/2) + O(n)`	mergesort, quicksort	`O(n log n)`
`T(n) = T(n-1) + O(n)`	selection or bubble sort	$O(n^2)$

60

Runtime complexity of mergesort?

Let N = r-l, the number of elements to sort

T(N) = …

```
34  public static void mergeHelper(int[] ar, int l, int r) {
35      int diff = r-l;
36      if (diff < 2) { return; }
37      int mid = l + diff/2;
38      mergeHelper(ar, l, mid);
39      mergeHelper(ar, mid, r);
40      merge(ar, l, mid, r);
41  }
```

T(N/2)+ …

T(N/2)+ …

O(N)

61