L15: Mergesort & Binary Search

Alex Steiger

CompSci 201: Spring 2024

3/4/2024

1

Announcements, Coming up

- Today, Monday 3/4
  - Project P3: DNA (linked list project) due
  - Project P4: Autocomplete out by tomorrow
- Wednesday 3/6
  - APT 6 (sorting problems) due
- Friday 3/8
  - Fill out the **midsemester course survey**
  - **No discussion, enjoy spring break!**
- Wednesday 3/20
  - Midterm 2, ~linked list through today

2

Midsemester Survey

- >70% submitted?
  - 1 extra credit pt on Exam 2
- >80% submitted?
  - 2 extra credit pts on Exam 2
- **Due Friday, 3/8!**

3

Today's Agenda

1. Sorting algorithms
   - Selection sort, mergesort
2. Binary search algorithm
3. Introduce Stack, Queue, PriorityQueue

4

Efficient sorting algorithms

See example implementations here

5

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

6

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 $O(N^2)$

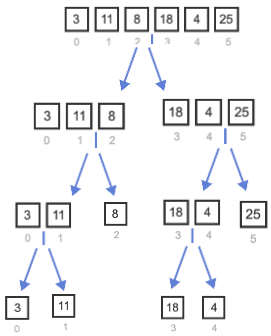
8
5
2
6
9
3
1
4
0
7

7

Mergesort

High level idea:

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



Zybook

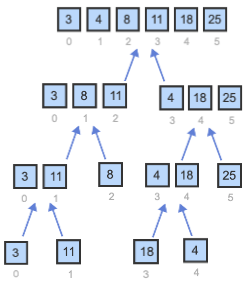
8

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

9

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

10

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

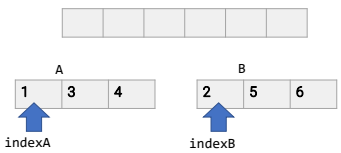
Recursively sort 2nd half

Merge the 2 sorted parts

11

Merge method concept

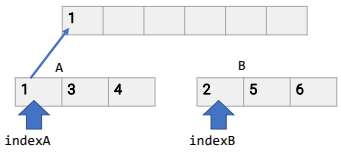
- Given two sorted arrays, **A** and **B**, want to merge them into one with all values from both.
- Need to keep track of **two** indices, **indexA** and **indexB**.



12

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`.



13

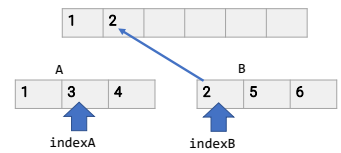
Merge method

- 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`.

14

Merge method

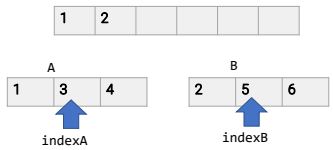
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



15

Merge method

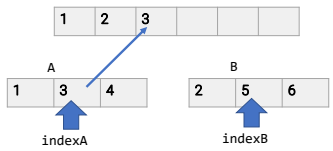
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



16

Merge method

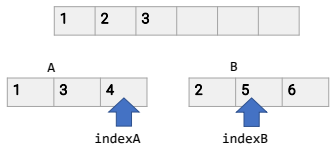
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



17

Merge method

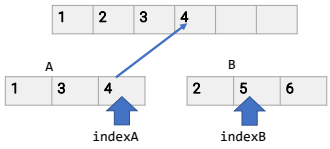
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



18

Merge method

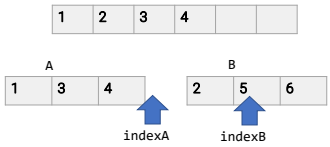
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



19

Merge method

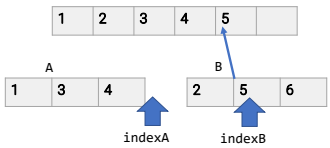
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



20

Merge method

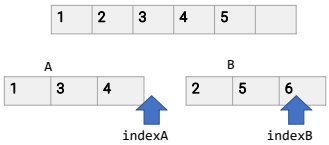
- Given two sorted arrays, `A` and `B`, want to merge them into one with all values from both.
- Need to keep track of **two** indices, `indexA` and `indexB`.



21

Merge method

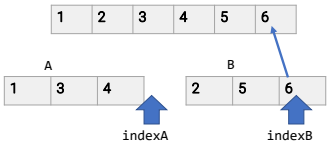
- Given two sorted arrays, **A** and **B**, want to merge them into one with all values from both.
- Need to keep track of **two** indices, **indexA** and **indexB**.



22

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**.



23

Merge method initialization

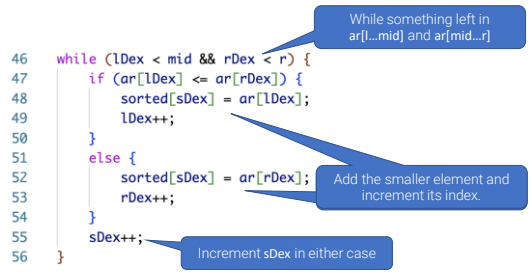
- Should merge **ar[l…mid]** and **ar[mid…r]**

```
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]**

24

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  }
```


25

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

26

Our implementation of mergesort used two methods shown below. Which method(s) are recursive? *

```
53  public static void mergeSort(int[] ar) {
54      mergeHelper(ar, l: 0, ar.length);
55  }
56
57  public static void mergeHelper(int[] ar, int l, int r) {
58      int diff = r-l;
59      if (diff < 2) { return; }
60      int mid = l + diff/2;
61      mergeHelper(ar, l, mid);
62      mergeHelper(ar, mid, r);
63      merge(ar, l, mid, r);
64  }
```

- ○ mergeSort and mergeHelper are both recursive
- ○ Only mergeSort is recursive
- ✓ Only mergeHelper is recursive
- ○ Neither are recursive

28

What best explains the purpose of the mergeSort *wrapper* method? *

```
53  public static void mergeSort(int[] ar) {
54      mergeHelper(ar, l: 0, ar.length);
55  }
56
57  public static void mergeHelper(int[] ar, int l, int r) {
58      int diff = r-l;
59      if (diff < 2) { return; }
60      int mid = l + diff/2;
61      mergeHelper(ar, l, mid);
62      mergeHelper(ar, mid, r);
63      merge(ar, l, mid, r);
64  }
```

- ◯ It helps us make the algorithm more efficient
- ◯ It helps us make the algorithm more correct
- ◯ It helps us avoid having to use recursion
- ✓ It helps us to initialize the parameters to the recursive calls

29

Based on what you see, how many *levels* of recursion will there be to the merge sort algorithm? To be precise: For a given index k of the original array, for how many recursive calls will k lie between l and r-1, where l and r are the parameters of the recursive call?

Answer in asymptotic notation as a function of N where N is the length of ar. *

```
53  public static void mergeSort(int[] ar) {
54      mergeHelper(ar, l: 0, ar.length);
55  }
56
57  public static void mergeHelper(int[] ar, int l, int r) {
58      int diff = r-l;
59      if (diff < 2) { return; }
60      int mid = l + diff/2;
61      mergeHelper(ar, l, mid);
62      mergeHelper(ar, mid, r);
63      merge(ar, l, mid, r);
64  }
```

- ◯ O(1)
- ✓ O(log(N))
- ◯ O(N)
- ◯ O(N log(N))

30

Let N = r-l. What is the asymptotic runtime complexity of the merge method? The runtime complexity of the arraycopy method is linear in the number of elements it copies, which is the last parameter of the method. *

```
66  public static void merge(int[] ar, int l, int mid, int r) {
67      int[] sorted = new int[r-l];
68      int sDex=0; int lDex=l; int rDex=mid;
69      while (lDex < mid && rDex < r) {
70          if (ar[lDex] <= ar[rDex]) {
71              sorted[sDex] = ar[lDex];
72              lDex++;
73          }
74          else {
75              sorted[sDex] = ar[rDex];
76              rDex++;
77          }
78          sDex++;
79      }
80      if (lDex == mid) { System.arraycopy(ar, rDex, sorted, sDex, r-rDex); }
81      else { System.arraycopy(ar, lDex, sorted, sDex, mid-lDex); }
82      System.arraycopy(sorted, srcPos: 0, ar, l, r-l);
83  }
```

- ◯ O(1)
- ◯ O(log(N))
- ✓ O(N)
- ◯ O(N log(N))
- ◯ O(N^2)

31

How fast is mergesort? Empirically…

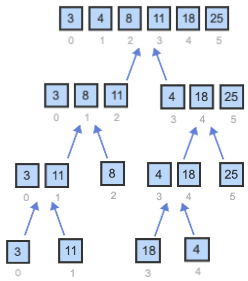
N (thousands)	Selection sort (ms)	Insertion sort (ms)	Merge sort (ms)	Java.util Arrays.sort (ms)
10k	22	40	1	2
30k	168	334	2	2
90k	1481	967	7	6
270k	13175	8716	22	14

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

32

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

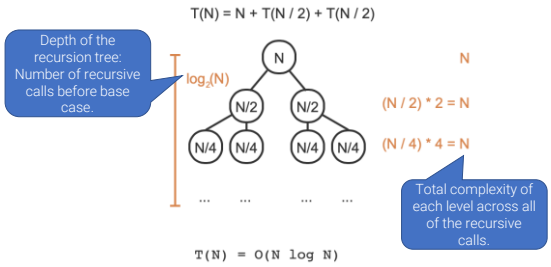
- Recursive subproblem ~halves in size.
  - How times can we halve before base case?
  - ~log N times ⇒ O(log N) ***levels of recursion***
- If we can do ALL of the merges at each ***level*** in O(N) total time?
- Overall [# levels]*O(N) = O(N log(N)) time



Zybook

33

Recursion tree



Visualization from the Zybook

34

Analyzing Recursive Runtime

Develop a recurrence relation of the form

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

- $T(N)$ - Total runtime on input size N
- $a \cdot T\big(g(N)\big)$ - Recursive call(s)
- $f(N)$ - Non-recursive runtime

Where:

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

(Not the most general formula, but enough for today/201)

35

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  }
```

$a = 1$: Only one rec. call

$g(N) = N - 1$: Rec. subprob. has list with one less node than input

$f(N) = O(1)$: O(1) ops, each O(1) time

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

Plugging in: $T(N) = T(N-1) + O(1)$

36

Solving Recurrence Relations

Total runtime

Apply recurrence again to T(N-1)

$$\begin{aligned} T(N) &= T(N-1) + 1 \\ &= (T(N-2) + 1) + 1 \\ &= \big((T(N-3) + 1) + 1\big) + 1 \\ &\vdots \\ &= T(1) + N \\ &= O(N) \end{aligned}$$

And again, to T(N-2)

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

37

Recurrence Relations and Expectations in 201

- In general, will ***not*** be asked to ***solve*** recurrence relations on exams (that's later in CS 230/330)
- You ***may*** 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	$O(N^2)$

38

Runtime complexity of mergesort?

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

```
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) = …

T(N/2)

T(N/2)

O(N)

$$T(N) = 2T\left(\frac{N}{2}\right) + O(N) \rightarrow T(N) \text{ is } O\big(N \log(N)\big)$$

39

Binary Search

40

Binary Search

- Given a ***sorted*** `list` of N elements and a `target` value, return:
  - Index `i` such that `list.get(i)` equals `target`, or
  - -1 if `target` not in `list`

- Example:
  - If we search for 'h', should return 4
  - If we search for 'c', should return -1

value	'a'	'b'	'd'	'g'	'h'	'j'	'k'	'm'	'p'
index	0	1	2	3	4	5	6	7	8

41

Java API Binary Search

`Arrays.binarySearch` (for arrays) and `Collections.binarySearch` (for Lists).

```
String[] ar = {"ape", "bird", "cat", "dog", "elephant", "ferret",
"gecko", "hippo"};
int index = Arrays.binarySearch(ar, "cat");
```

Returns 2

Careful, assumes input is sorted (and does not verify)!

```
String[] ar = {"cat", "ape", "bird",...
int index = Arrays.binarySearch(ar, "cat");
```

Returns -4

42

Java API Binary Search with Comparator

Can pass a comparator `comp`, in which case:

1. Array/List should be sorted by that `comp`, and
2. Want an index `i` with `i`'th element e_i has `comp.compare(`e_i`, target)==0`.

Sorted by length

[ape, cat, dog, bird, gecko, hippo, ferret, elephant]

```
Comparator<String> comp =
    Comparator.comparing(String::length);
index = Arrays.binarySearch(ar, "dog", comp);
```

Returns 1. `comp.compare("cat", "dog")==0`

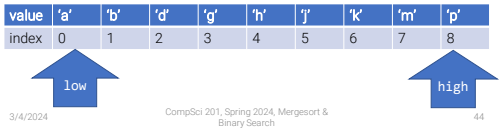
43

How is Binary Search O(log(N))?

- How to find something in a list of N elements without looping over the list?
- Let **`low`** (initially 0) and **`high`** (initially N-1) mark the limits of the active search space.
- Want to cut down the search space by half at each step:

N
N/2
N/4
N/8
…
1

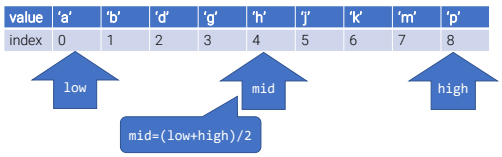
log_2(N) steps!



44

Binary Search in Pictures

- Searching for 'd' in

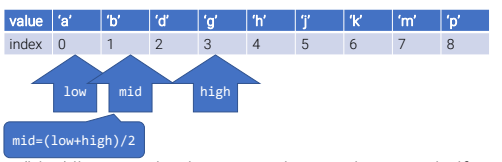


- 'h' > 'd', so need to keep searching in the ***lower*** half.
- Set **`high = mid-1;`**

45

Binary Search in Pictures

- Searching for 'd' in

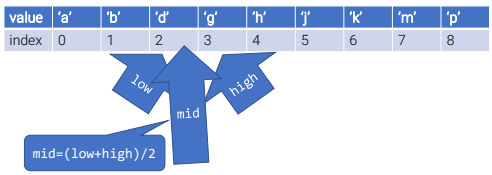


- 'b' < 'd', so need to keep searching in the ***upper*** half.
- Set **`low = mid+1;`**

46

Binary Search in Pictures

- Searching for 'd' in



- 'd' equals 'd', return `mid (2)`

47

Reasoning about Coding Binary Search

- Going to loop `while (low <= high)`
  - Looping while there is anything left to search
- For correctness, want to maintain the following **loop invariant:**
  - If the target is in the array/list, it is in the range [low, high]
- At each step, either find the target and return, or…cut [low, high] in half without losing the target
  - Needs sortedness

48

Iterative Code for DIY Binary Search?

```
7  public static <T> int binarySearch(List<T> list, T target, Comparator<T> comp) {
8      int low = 0;
9      int high = list.size()-1;
10     while (low <= high) {
11         int mid = (low + high)/2;
12         T midval = list.get(mid);
13
14         int cmp = comp.compare(midval,target);
15         if (cmp < 0)
16             low = mid + 1;
17         else if (cmp > 0)
18             high = mid - 1;
19         else
20             return mid; // target found
21     }
22     return -1;  // target not found
23 }
```

<T> for generic type, can be a String list, Integer list, …, just need `target` and `Comparator` of the same type.

49