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

Midsemester Survey

• >70% submitted?
  • 1 extra credit pt on Exam 2

• >80% submitted?
  • 2 extra credit pts on Exam 2

• Due Friday, 3/8!
Today’s Agenda

1. Sorting algorithms
   • Selection sort, mergesort

2. Binary search algorithm

3. Introduce Stack, Queue, PriorityQueue

Efficient sorting algorithms

See example implementations here

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.
Selection Sort Code and Runtime

```java
public static void selectSort(int[] arr) {
    for (int i=0; i<arr.length; i++) {
        int minIdx = i;
        for (int j=i+1; j<arr.length; j++) {
            if (arr[j] < arr[minIdx]) {
                minIdx = j;
            }
        }
        int temp = arr[i];
        arr[i] = arr[minIdx];
        arr[minIdx] = temp;
    }
}
```

Mergesort

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

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

Mergesort recursive method

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

```java
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, r);
41 }
```

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`.
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.
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.
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.

```
1 2 3 4
A
1 3 4

1 3 4

2 5 6
B

2 5 6

IndexA

IndexB
```

```
1 2 3 4
A
1 3 4

1 3 4

2 5 6
B

2 5 6

IndexA

IndexB
```
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.

Merge method initialization

• Should merge \( A[l...mid] \) and \( A[mid...r] \)

```java
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 \( ar[mid...r] \)
Merge method loop

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

Add the smaller element and increment its index.

Increment sDex in either case

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, 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

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

- Only mergeSort is recursive
- Only mergeHelper is recursive
- Neither are recursive
Let N = nL. 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. *

- O(1)
- O(log N)
- O(N)
- O(N log N)
- O(N^2)
- O(N^3)

Based on what you see, how many levels of recursion will there be in the merge sort algorithm? To be precise for a given index k of the original array, for how many recursive calls will k be in between [0 + k, 1), where 0 and 1 are the parameters of the recursive call?

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

- O(1)
- O(log N)
- O(N)
- O(N log N)
- O(N^2)
- O(N^3)

What best explains the purpose of the mergeSort wrapper? *

- 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 call.
How fast is mergesort? Empirically...

<table>
<thead>
<tr>
<th>N (thousands)</th>
<th>Selection sort (ms)</th>
<th>Insertion sort (ms)</th>
<th>Merge sort (ms)</th>
<th>Java.util.Arrays.sort (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10k</td>
<td>22</td>
<td>40</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>30k</td>
<td>168</td>
<td>334</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>90k</td>
<td>1481</td>
<td>967</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>270k</td>
<td>13175</td>
<td>8716</td>
<td>22</td>
<td>14</td>
</tr>
</tbody>
</table>

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

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

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

Recursion tree

$T(N) = N + T(N/2) + T(N/2)$

Depth of the recursion tree: Number of recursive calls before base case

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

Visualization from the Zybook
Analyzing Recursive Runtime

Develop a recurrence relation of the form
\[ T(N) = a \cdot T(g(N)) + f(N) \]

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)

Analyzing Runtime of Recursive Reverse

```java
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: \( T(N) = a \cdot T(g(N)) + f(N) \)
Plugging in: \( T(N) = T(N - 1) + O(1) \)

Solving Recurrence Relations

\[ T(N) = T(N - 1) + 1 = (T(N - 2) + 1) + 1 = \ldots = (T(N - 3) + 1) + 1 \]
\[ = \ldots = T(1) + N = O(N) \]

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

<table>
<thead>
<tr>
<th>Recurrence</th>
<th>Algorithm</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(N) = T\left(\frac{N}{2}\right) + O(1) )</td>
<td>binary search</td>
<td>( O(\log N) )</td>
</tr>
<tr>
<td>( T(N) = T(N-1) + O(1) )</td>
<td>sequential search</td>
<td>( O(N) )</td>
</tr>
<tr>
<td>( T(N) = 2T\left(\frac{N}{2}\right) + O(1) )</td>
<td>tree traversal</td>
<td>( O(N) )</td>
</tr>
<tr>
<td>( T(N) = T\left(\frac{N}{2}\right) + O(N) )</td>
<td>array partition, find ( k^{th} )</td>
<td>( O(N) )</td>
</tr>
<tr>
<td>( T(N) = 2T\left(\frac{N}{2}\right) + O(1) )</td>
<td>mergesort, quicksort</td>
<td>( O(N \ log N) )</td>
</tr>
<tr>
<td>( T(N) = T(N-1) + O(N) )</td>
<td>selection or bubble sort</td>
<td>( O(N^2) )</td>
</tr>
</tbody>
</table>

Runtime complexity of mergesort?

Let \( N = r - 1 \), the number of elements to sort

\[
T(N) = 2T\left(\frac{N}{2}\right) + O(N) \quad \text{is} \quad O(N \log(N))
\]

Binary Search
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

<table>
<thead>
<tr>
<th>value</th>
<th>y'</th>
<th>2'</th>
<th>df</th>
<th>y'</th>
<th>h'</th>
<th>T</th>
<th>x'</th>
<th>y'</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

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");` \( \rightarrow \text{Returns 2} \)
  - Careful, assumes input is sorted (and does not verify)!  
  - `String[] ar = {"cat", "ape", "bird", ...};`
  - `int index = Arrays.binarySearch(ar, "cat");` \( \rightarrow \text{Returns -4} \)

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
     \( \text{comp.compare}(e_i, \text{target})==0 \).

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

Returns 1
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:

<table>
<thead>
<tr>
<th>value</th>
<th>'a'</th>
<th>'b'</th>
<th>'d'</th>
<th>'g'</th>
<th>'h'</th>
<th>'j'</th>
<th>'k'</th>
<th>'m'</th>
<th>'p'</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Binary Search in Pictures

• Searching for 'd' in

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

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

<table>
<thead>
<tr>
<th>value</th>
<th>'a'</th>
<th>'b'</th>
<th>'d'</th>
<th>'g'</th>
<th>'h'</th>
<th>'j'</th>
<th>'k'</th>
<th>'m'</th>
<th>'p'</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

Binary Search in Pictures

• Searching for 'd' in

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

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

mid = (low + high) / 2

<table>
<thead>
<tr>
<th>value</th>
<th>'a'</th>
<th>'b'</th>
<th>'d'</th>
<th>'g'</th>
<th>'h'</th>
<th>'j'</th>
<th>'k'</th>
<th>'m'</th>
<th>'p'</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Binary Search in Pictures

- Searching for 'd' in

<table>
<thead>
<tr>
<th>Value</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

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

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

Iterative Code for DIY Binary Search?

```java
public static <T> int binarySearch(List<T> list, T target, Comparator<T> cmp) {
    int low = 0;
    int high = list.size() - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        T midVal = list.get(mid);
        if (cmp.compare(midVal, target) < 0) {
            low = mid + 1;
        } else if (cmp.compare(midVal, target) > 0) {
            high = mid - 1;
        } else { // target found
            return mid;
        }
    }
    return -1; // target not found
}
```