L15: Mergesort & Binary Search

Alex Steiger
CompSci 201: Spring 2024
3/4/2024
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[] ar) {
    for (int i=0; i<ar.length; i++) {
        int minDex = i;
        for (int j=i+1; j<ar.length; j++) {
            if (ar[j] < ar[minDex]) {
                minDex = j;
            }
        }
        int temp = ar[i];
        ar[i] = ar[minDex];
        ar[minDex] = temp;
    }
}
```

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

High level idea:
• Base case: size 1
  • Return list
• Recursive case:
  • Mergesort(first half)
  • Mergesort(second half)
  • ...
Mergesort

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

3/4/2024
CompSci 201, Spring 2024, Mergesort & Binary Search
Zybook
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, l: 0, ar.length);
32 }
```

Want to specify a left and right boundary of the subarray for each recursive call to sort
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, 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
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, \( \text{index}_A \) and \( \text{index}_B \).
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, \( \text{indexA} \) and \( \text{indexB} \).
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.

![Diagram showing the merge process between two sorted arrays A and B, with indices indexA and indexB.](image-url)
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.

```
A: 1 3 4
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, \( \text{indexA} \) and \( \text{indexB} \).
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 $ar[l...mid]$ and $ar[mid...r]$ 

```java
public static void merge(int[] ar, int l, int mid, int r) {
  int[] sorted = new int[r-l];
  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

```cpp
while (lDex < mid && rDex < r) {
    if (ar[lDex] <= ar[rDex]) {
        sorted[sDex] = ar[lDex];
        lDex++;
    }
    else {
        sorted[sDex] = ar[rDex];
        rDex++;
    }
    sDex++;
}
```

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 \texttt{ar[l...mid]} or \texttt{ar[mid...r]} first, need to copy the rest of the other.

• Then need to copy sorted back onto \texttt{ar[l...r]}

\begin{verbatim}
if (lDex == mid) {
    System.arraycopy(ar, rDex, sorted, sDex, r-rDex);
} else {
    System.arraycopy(ar, lDex, sorted, sDex, mid-lDex);
}
System.arraycopy(sorted, srcPos: 0, ar, l, r-l);
\end{verbatim}

• Code uses the \texttt{System.arraycopy} method:

\begin{verbatim}
public static void arraycopy(Object src,
    int srcPos,
    Object dest,
    int destPos,
    int length)
\end{verbatim}

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?

- mergeSort and mergeHelper are both recursive
- Only mergeSort is recursive
- Only mergeHelper is recursive
- Neither are recursive
What best explains the purpose of the mergeSort wrapper method? *

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

public static void mergeHelper(int[] ar, int l, int r) {
    int diff = r-l;
    if (diff < 2) { return; }
    int mid = l + diff/2;
    mergeHelper(ar, l, mid);
    mergeHelper(ar, mid, r);
    merge(ar, l, mid, r);
}
```

- 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
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 \( ar \). *
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. * 

```java
public static void merge(int[] ar, int l, int mid, int r) {
    int[] sorted = new int[r-l];
    int sDex=0; int lDex=l; int rDex=mid;
    while (lDex < mid && rDex < r) {
        if (ar[lDex] <= ar[rDex]) {
            sorted[sDex] = ar[lDex];
            lDex++;
        } else {
            sorted[sDex] = ar[rDex];
            rDex++;
        }
        sDex++;
    }
    if (lDex == mid) { System.arraycopy(ar, rDex, sorted, sDex, r-rDex); } else { System.arraycopy(ar, lDex, sorted, sDex, mid-lDex); }
    System.arraycopy(sorted, srcPos: 0, ar, l, r-l);
}
```
How fast is mergesort? Empirically...

<table>
<thead>
<tr>
<th>N (thousands)</th>
<th>Selection sort (ms)</th>
<th>Insertion sort (ms)</th>
<th>Mergesort (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?
  - ~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
Recursion tree

\[ T(N) = N + T(N/2) + T(N/2) \]

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

Total complexity of each level across all of the recursive calls.

\[ T(N) = O(N \log N) \]

Visualization from the Zybook
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

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

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

\( a = 1 \): 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
Solving Recurrence Relations

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

- Total runtime
- Apply recurrence again to \( T(N-1) \)
- And again, to \( T(N-2) \)
- \( T(1) \) 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>$T(N) = T(N/2) + O(1)$</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>$T(N) = 2T(N/2) + O(1)$</td>
<td>tree traversal</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>$T(N) = T(N/2) + O(N)$</td>
<td>qsort partition, find $k^{th}$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>$T(N) = 2T(N/2) + O(N)$</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 \( N = r - l \), the number of elements to sort

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

• Given a sorted list of N elements and a target value, return:
  • Index $i$ such that $\text{list.get}(i)$ 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>‘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>
Java API Binary Search

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

```java
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)!

```java
String[] ar = {"cat", "ape", "bird",...}

int index = Arrays.binarySearch(ar, "cat");

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 `comp.compare(e_i, target)==0`.

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

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

Sorted by length

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

$N$
$N/2$
$N/4$
$N/8$

$...$
$1$

$log_2(N)$ steps!
Binary Search in Pictures

- Searching for ‘d’ in

<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>

- ‘h’ > ‘d’, so need to keep searching in the lower half.
- Set high = mid - 1;
Binary Search in Pictures

• Searching for ‘d’ in

<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>

- ‘b’ < ‘d’, so need to keep searching in the upper half.
- Set low = mid+1;
Binary Search in Pictures

- Searching for ‘d’ in

<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>

\[
\text{mid} = \frac{\text{low} + \text{high}}{2}
\]

- ‘d’ equals ‘d’, return \text{mid} (2)
Reasoning about Coding Binary Search

• Going to loop \texttt{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> comp) {
    int low = 0;
    int high = list.size()-1;
    while (low <= high) {
        int mid = (low + high)/2;
        T midval = list.get(mid);

        int cmp = comp.compare(midval, target);
        if (cmp < 0)
            low = mid + 1;
        else if (cmp > 0)
            high = mid - 1;
        else
            return mid; // target found
    }
    return -1; // target not found
}
```

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