CSCI 241

Lecture 9
Runtime of Quick, Merge, and Radix
How's A1 going?
Announcements

• Feedback survey out, please submit by Monday

• Quiz today: same as usual

• In-class problems for today are posted on the course webpage (schedule table, 4/24):
  https://facultyweb.cs.wwu.edu/~wehrwes/courses/csci241_20s/lectures/L09/sort_runtimes.html

  • Please pull them up so you can refer to them while in breakout rooms.
Goals

• Get more practice analyzing runtimes.

• Know how logarithms end up in runtime counts.

• Know the runtime complexity of all the sorting algorithms we’ve covered.
Asymptotic Runtime Class
or, "big-O" runtime

1. Count the number of primitive (constant-time) operations that occur over the entire execution of the algorithm.

2. Drop constants and lower-order terms to find the asymptotic runtime class.

- Tells us how the runtime grows as the input size grows.
- Doesn't tell us anything about runtime when the input is small!
0. Warmup:

What's the runtime of `mins`?

```java
/** Return the max value in A[start..end] */
public int findMax(int[] a, int start, int end) {
    int currentMax = a[start];
    for (int i = start + 1; i < end; i++) {
        if (currentMax < a[i]) {
            currentMax = a[i];
        }
    }
    return currentMax;
}

/** Print the min of several subarrays of A
 * Precondition: A.length >= 50. */
public static void mins(A) {
    for (int i = 1; i < 50; i++) {
        System.out.println(findMin(A, 0, i));
    }
}
```

Runtime Analysis:
- `findMax` has a time complexity of $O(n)$
- `mins` has a time complexity of $O(n^2)$

Total time complexity of `mins` is $O(n^2)$, considering the nested loop.
2. Something new...

```java
public int f(int n) {
    while (n > 0) {
        System.out.println(n);
        n = n/2;
    }
}
```

How many times can \( n \) be divided by 2 before becoming 0?

\[
\log_2 n = x
\]

\[
\frac{n}{2^x} = 1 \quad n = 2^x
\]

\[
\log_2 n = x
\]

\[
\log_2 n = k \log_10 n
\]
Recursive methods:

1. How much work is actually done per call? 
   *not counting the recursive calls*

2. How many calls are made?
   - This is simpler when the work per call is the same.
   - Sometimes the work per call depends on n.
/** sort A[start..end] using mergesort */

mergeSort(A, start, end):
if (end-start < 2):
    return mid = (end+start)/2
mergeSort(A,start,mid)
mergeSort(A,mid, end)
merge(A, start, mid, end)

1. How much work is actually done per call?
Merge step: Loop Invariant

Set \( i = j \)

Make new array \( A \)

While neither section is empty:

Copy smaller of \( A[i:mid], A[j:] \)

Into \( B \)

Increment \( i, j \) or \( k \)

Copy remaining values from left or right half

Postcondition:

- Sorted

- Merged
2. Runtime of merge

initialize i, j
B = deep copy of A
while neither uncopied segment is empty:
    copy the smaller of B[i], B[j] into A[k]
    increment i or j
    increment k

while one uncopied segment is empty:
    copy the next element in the nonempty segment into A[k]
    increment i or j
    increment k
2. Runtime of merge

initialize $i, j$  
$B = \text{deep copy of } A$  
while neither uncopied segment is empty:
  copy the smaller of $B[i], B[j]$ into $A[k]$  
  increment $i$ or $j$  
  increment $k$

while one uncopied segment is empty:
  copy the next element in the nonempty segment into $A[k]$  
  increment $i$ or $j$  
  increment $k$

Invariant

$\mathbf{A}$  
merged  
$k$  

$\mathbf{B}$  
not yet copied  
not yet copied  
copied  
copied

$1 + O(n) + O(n) + O(n) = \mathcal{O}(n)$
Runtime Analysis: MergeSort

/** sort A[start..end] using mergesort */
mergeSort(A, start, end):
    if (end-start < 2):
        return mid = (end+start)/2
mergeSort(A,start,mid) O(?)
mergeSort(A,mid, end) O(?)
merge(A, start, mid, end) O(n)

1. How much work is actually done per call? O(n)
Runtime Analysis: MergeSort

```c
/** sort A[start..end] using mergesort */
mergeSort(A, start, end):
    if (end-start < 2):
        return mid = (end+start)/2  O(1)
mergeSort(A, start, mid)  O(?)
mergeSort(A, mid, end)   O(?)
merge(A, start, mid, end)  O(n)
```

2. How many calls are made?
How many calls to mergesort?

\[ O(n \log n) \]