Recitation 9

Analysis of Algorithms and Inductive Proofs

Review: Big O definition

\[ f(n) \text{ is } O(g(n)) \]

iff there exists \( c > 0 \) and \( N > 0 \) such that:
\[ f(n) \leq c \cdot g(n) \text{ for } n \geq N \]

Example: \( n+6 \) is \( O(n) \)

\[
\begin{align*}
\text{n + 6} & \quad \text{---this is } f(n) \\
\leq & \quad \text{<if } 6 \leq n, \text{ write as}> \\
\text{n} & \quad \text{<arith>} \\
\text{=} & \quad \text{choose } c = 2 \\
2^n & \quad \text{---this is } c \cdot g(n) \\
\text{So choose } c = 2 \text{ and } N = 6
\end{align*}
\]

Review: Big O

1. \( \log(n) + 20 \) is \( O(\log(n)) \) (logarithmic)
2. \( n + \log(n) \) is \( O(n) \) (linear)
3. \( n^2 \) and \( 3^n \) are \( O(n^2) \) (quadratic)
4. \( n \cdot \log(n) + n \) is \( O(n \cdot \log(n)) \)
5. \( n^2 + 2n + 6 \) is \( O(n^2) \) (quadratic)
6. \( n^2 + n^2 \) is \( O(n^2) \) (cubic)
7. \( 2^n + n^5 \) is \( O(2^n) \) (exponential)

Merge Sort
Merge Sort

Runtime of merge sort

/** Sort b[h..k]. */
public static void mS(Comparable[] b, int h, int k) {
    if (h >= k) return;
    int e = (h+k)/2;
    mS(b, h, e);
    mS(b, e+1, k);
    merge(b, h, e, k);
}

mS is mergeSort for readability

---

Merge Sort

Runtime of merge sort

/** Sort b[h..k]. */
public static void mS(Comparable[] b, int h, int k) {
    if (h >= k) return;
    int e = (h+k)/2;
    mS(b, h, e);
    mS(b, e+1, k);
    merge(b, h, e, k);
}

Recursive Case:

T(n) = 2T(n/2) + O(n)

---

Merge Sort

Runtime of merge sort

/** Sort b[h..k]. */
public static void mS(Comparable[] b, int h, int k) {
    if (h >= k) return;
    int e = (h+k)/2;
    mS(b, h, e);
    mS(b, e+1, k);
    merge(b, h, e, k);
}

How long does merge take?

- We will count the number of comparisons mS makes
- Use T(n) for the number of array element comparisons that mS makes on an array segment of size n

---

Merge Sort

Runtime of merge sort

/** Sort b[h..k]. */
public static void mS(Comparable[] b, int h, int k) {
    if (h >= k) return;
    int e = (h+k)/2;
    mS(b, h, e);
    mS(b, e+1, k);
    merge(b, h, e, k);
}

Recursive Case:

T(n) = 2T(n/2) + O(n)
Runtime

We determined that
\[ T(1) = 0 \]
\[ T(n) = 2T(n/2) + n \quad \text{for } n > 1 \]

We will prove that
\[ T(n) = n \log_2 n \quad \text{(or } n \log n \text{ for short)} \]

Proof by induction

To prove \( T(n) = n \log n \),
we can assume true for smaller values of \( n \) (like recursion)

\[ T(n) = 2T(n/2) + n \]
\[ = 2(n/2)\log(n/2) + n \]
\[ = n(\log n - 1) + n \]
\[ = n \log n \]

Property of logarithms
\[ \log_2 2 = 1 \]

Heap Sort

Very simple idea:
1. Turn the array into a max-heap
2. Pull each element out

```java
/** Sort b */
public static void heapSort(Comparable[] b) {
    heapify(b);
    for (int i = b.length - 1; i >= 0; i--)
        b[i] = poll(b, i);
}
```

Why does it have to be a max-heap?
/** Sort b */
public static void heapSort(Comparable[] b) {
    heapify(b);
    for (int i = b.length-1; i >= 0; i--)
        b[i] = poll(b, i);
}

Heap Sort runtime

Total runtime:
O(n lg n) + n*O(lg n) = O(n lg n)