Recitation 9

Analysis of Algorithms and Inductive Proofs

Review: Big O definition

\[ f(n) \leq c \cdot g(n) \text{ for } n \geq N \]

There exists \( c > 0 \) and \( N > 0 \) such that:

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

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

Review: Big O

Is used to classify algorithms by how they respond to changes in input size \( n \).

Important vocabulary:
- Constant time: \( O(1) \)
- Logarithmic time: \( O(\log n) \)
- Linear time: \( O(n) \)
- Quadratic time: \( O(n^2) \)
- Exponential time: \( O(2^n) \)

Merge Sort
Runtime of merge sort

```java
/** 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
```

● 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 \( n \)

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

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

Proof by induction
To prove \( T(n) = n \log_2 n \), we can assume true for smaller values of \( n \) (like recursion)
\[ T(n) = 2T(n/2) + n \]
\[ = 2(n/2) \log(n/2) + n \]
\[ = n(\log n - \log 2) + n \]
\[ = n\log n - n + n \]
\[ = n\log n \]

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?
Heap Sort runtime

```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);
    }
}
```

Heap Sort runtime:
- \(O(n \lg n)\)
- loops \(n\) times

Total runtime:
\(O(n \lg n) + n \cdot O(\lg n) = O(n \lg n)\)