CS61B Spring 2016 Secret Section 2 Worksheet

CS61B Tutors

Week 2

1 Big O Ordering

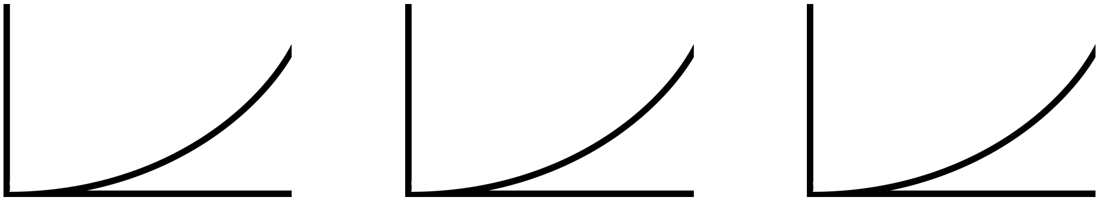
Rank the following from smallest to largest growths:

$O(\sqrt{n})$ $O(\log n)$ $O(2^n)$
$O(e^n)$ $O(n^{3/2})$ $O(n \log n)$
$O(1)$ $O(\log^2 n)$ $O(n!)$
$O(n^n)$ $O(n)$

Solution: $O(1) < O(\log n) < O(\log^2 n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^{3/2}) < O(2^n) < O(e^n) < O(n!) < O(n^n)$

2 Warmup

Starting from the graph on the left, shade in the regions that correspond to $O(n^2), \Omega(n^2), \Theta(n^2)$, respectively.



Solution: First graph shows $O(n^2)$, so color in the under side of the graph; Second graph shows $\Omega(n^2)$, so color in the top side of the graph; Third graph shows $\Theta(n^2)$, so color in the line itself

3 Big O Notation

Find the tightest O, Ω, Θ functions that bound the following:

1. $5n + 6 - 3n$ Solution: $\Theta(n)$
2. $2^n + 2^{n-1}$ Solution: $\Theta(2^n)$
3. $n^2 + n \log n + 3n$ Solution: $\Theta(n^2)$
4. $\log n + \log(n^2)$ Solution: $\Theta(\log(n^2))$
5. $\log n!$ Solution: $\Theta(n \log(n))$
6. $1 + 2 + ... + n$ Solution: $\Theta(n^2)$

4 Runtime Analysis

What are the O, Ω, Θ runtimes of the following function?

```
double minDistance = point[0].distance(point[1]);

  /* Visit a pair (i, j) of points. */
  for (int i = 0; i < numPoints; i++) {
    /* We require that j > i so that each pair is visited only once. */
    for (int j = i + 1; j < numPoints; j++) {
      double thisDistance = point[i].distance(point[j]);
      if (thisDistance < minDistance) {
        minDistance = thisDistance;
      }
    }
  }
```

General solution:
$\Theta(n^2)$
This comes from the summation of the loops. First we do the inner loop 1 time, then 2, then 3, for a summation of: $1 + 2 + 3 + \cdots + n$ This is equal to $n(n-1)/2$
Please keep in mind that other solutions work when using O or Ω notation

5 More Runtime Analysis

What are the best case and worst case O, Ω, Θ runtimes of the following contrived function?

```
//runs in O(n) time
public static void linear(){...}
//runs in O(n^2) time
public static void squared(){...}
//runs in O(n^4) time
public static void fourth(){...}
//runs in O(n^5) time
public static void fifth(){...}

public static void contrived(n){
  if (n % 2 == 0){
    if (Math.random() > 0.5){
      linear();
    } else {
      squared();
    }
  } else {
    if (Math.random() > 0.5){
      fourth();
    } else {
      fifth();
    }
  }
}
```

General solution:
Best case: $\Theta(n)$
Worst case: $\Theta(n^5)$
Please keep in mind that other solutions work when using O or Ω notation

6 Even More Runtime Analysis

Assume sortedList is a sorted list of length n with no duplicates. What is the running time of the function useless? What does it print?

```java
static void useless(int[] sortedList) {
  for (int i = 0; i < sortedList.length; i++) {
    System.out.println(foo(sortedList, sortedList[i]));
  }
}

static int foo(int[] lst, int toFind) {
  return bar(lst, toFind, 0, lst.length);
}

static int bar(int[] lst, int toFind, int lower, int upper) {
  if (lower == upper) {
    return -1;
  }
  int mid = (lower + upper) / 2;
  if (lst[mid] > toFind) {
    return bar(lst, toFind, lower, mid);
  } else if (lst[mid] < toFind) {
    return bar(lst, toFind, mid + 1, upper);
  }
  return mid;
}
```

General solution:
$\Theta(n \log(n))$ as we are performing a binary search on every value in the array. The binary search takes $\Theta(\log(n))$ time, and we do this n times, for a total of $\Theta(n \log(n))$

7 Designing Algorithms

Write a function that determines if an array has all unique characters in $O(n^2)$ time.

```java
public static boolean hasUniqueCharacters(char[] characters){
  for (int i = 0; i < characters.length; i++) {
    for (int j = 0; j < characters.length; j++) {
      if (i != j) {
        if (characters[i] == characters[j]) {
          return false;
        }
      }
    }
  }
  return true;
}
```

Now try to do it in $O(n)$ time. Assume the only characters are lowercase a-z, 0-9.

```java
public static boolean hasUniqueCharacters(char[] characters){
  boolean[] beenSeenBefore = new char[256];
  for (int i = 0; i < characters.length; i++) {
    if (beenSeenBefore[(int) characters[i]] == true) {
      return false;
    }
    beenSeenBefore[(int) characters[i]] = true;
  }
  return true;
}
```