Sagar Desai 7/20/18

Analysis of Algorithms

Exercise I. *Give an analysis of the running time with respect to the input size n of each of the following program fragments below:*

a)
```
a = 0;
while (a < n * n * n)
  a = a + n * n;
```
$O(n^3/n^2) = O(n)$

b)
```
// input array is of length n
i = array.length - 1;
while (array[i] > x && i >= 0)
   i = i/2;
```
$O(\log_2 n)$

c)
```
sum = 0;
for (i = 0; i < Math.sqrt(n) / 2; i++)
  for ( j = i; j < 8 + i; j++)
    for (k = j; k < 8 + j; k++)
       sum++;
```
$O(\sqrt{n})$

d)
```
sum = 0;
for (i = 1; i < n; i *= 2)
   for (j = 0; j < n; j++)
      sum++;
```
$O(n \log n)$

e)
```
sum = 0;
for (i = 0; i < n; i++)  1
  for (j = i + 1; j < n; j++)  2
    for (k = j + 1; k < n; k++)  3
      for (l = k + 1; l < 10 + k; l++)
        sum++;
```
$O(n^3)$

f)
```
bunnyEars = function (bunnies) { // here bunnies === n
  if (bunnies === 0) return 0;
  return 2 + bunnyEars(bunnies-1);  1
}
```
$O(n)$

g)
```
search = function (array, arraySize, target) { // here arraySize === n
  if (arraySize > 0) {
    if (array[arraySize-1] === target) return true;
    else return search(array, arraySize-1, target);
  }
 return false;
}
```
$O(n)$

```
function maxD(arr){
      let min = arr[0];
      let maxD = 0;
      arr.forEach(val => {
            min = Math.min(min, val);
            maxD = Math.max(maxD, val-min);
            });
      return maxD;
```

Exercise II.

a) Given an array a of n numbers, design a linear running time algorithm to find the maximum value of a[j] - a[i], where $j \geq i$.

b) Suppose that you have an n-story building and plenty of eggs. Suppose also that an egg is broken if it is thrown off floor f or higher, and unbroken otherwise. Devise a strategy to determine the value of f such that the number of dropped eggs is minimized. Binary search for a height at which an egg no longer breaks. You can start at any height but at n/2 for efficieny

Exercise III. Below is the the pseudo-code for the Quicksort algorithm:

```
function quicksort(array)
   if length(array) <= 1
       return array
   select and remove a pivot element pivot from array
   create empty lists less and greater
   for each x in array
       if x <= pivot then append x to less
       else append x to greater
   return concatenate(quicksort(less), list(pivot), quicksort(greater))
```

a) Suppose we implement quicksort so that the pivot is always chosen to be the first element of the array. What is the running time of this algorithm on an input array that is already sorted? Why? $O(n^2)$

b) Suppose we implement quicksort so that the pivot is always magically chosen to be the median element of the array. What is the running time of this algorithm? Why? $O(n\log n)$