Analysis and Design of Algorithms Homework 2

Arturo Fornés Arvayo A01227071

February 24, 2017

1. Cormen Problems

   (a) 5.4.1 - How many people must there be in a room before the probability that someone has the same birthday as you do is at least $\frac{1}{2}$? How many people must there be before the probability that at least two people have a birthday on July 4 is greater than $\frac{1}{2}$?

      i. Assuming we have a uniform distribution: the probablity that at least someone has the same birthday as you in a group k of people can be defined as the complement of k people not having the same birthday as you:

         P{Given a day, someone is born that day} $= \frac{1}{n}$, where n is the number of days in a year (I'll round it up to 365).
         P{A person having a birthday different from you} $= 1 - \frac{1}{n} = \frac{n-1}{n}$
         Then P{k people having different birthdays from you} $= (\frac{n-1}{n})^k$
         And its complement will be P{Someone having the same birthday as you} $= 1 - (\frac{n-1}{n})^k$
         We bound it to be at least $\frac{1}{2}$ and we get:

$$1 - (\frac{n-1}{n})^k \geq \frac{1}{2}$$

$$-(\frac{n-1}{n})^k \geq -\frac{1}{2}$$

$$(\frac{n-1}{n})^k \leq \frac{1}{2}$$

$$klg(\frac{n-1}{n}) \leq lg(\frac{1}{2})$$

$$k \leq \frac{lg(\frac{1}{2})}{lg(\frac{n-1}{n})}$$

$$k \leq \frac{lg(1) - lg(2)}{lg(\frac{364}{365})}$$

$$k \leq \frac{-1}{lg(364) - lg(365)}$$

$$k \leq 252.652$$

k is then $\simeq 252$

ii. For having at least to people be born on the same date, a binary distribution can be used to model the distribution of birthdays (as birthday dates are independent with P{being born on a date} $= \frac{1}{n}$) and we get the probability of 0 people being born on the 4th of July and of 1 person and subtract it from 1 to get the complement (bound the addition to be greater than $\frac{1}{2}$):

$$\frac{1}{2} < 1 - P(1\,person) - P(0\,persons)$$

$$\frac{1}{2} > P(1\,person) + P(0\,persons)$$

$$P(0) = \binom{k}{0}(\frac{1}{n})^0(\frac{n-1}{n})^k = (\frac{n-1}{n})^k$$

$$P(1) = \binom{k}{1}(\frac{1}{n})^1(\frac{n-1}{n})^{k-1} = \frac{k}{n}(\frac{n-1}{n})^{k-1}$$

$$\frac{1}{2} > (\frac{k}{n})(\frac{n-1}{n})^{k-1} + (\frac{n-1}{n})^k$$

$$\frac{1}{2} > (\frac{n-1}{n})^{k-1}(\frac{k}{n} + (\frac{n-1}{n})^{k-(k-1)})$$

$$\frac{1}{2} > (\frac{n-1}{n})^{k-1}(\frac{k+n-1}{n})$$

$$\frac{1}{2} > (\frac{364}{365})^{k-1}(\frac{k+364}{365})$$

Using the included Python script *birthdayEstimate.py* we get that the probability of at least two people to be born in the 4th of July is greater than $\frac{1}{2}$ for $k \geq 613$.

(b) 5.4.6 - Suppose that n balls are tossed into n bins, where each toss is independent and the ball is equally likely to end up in any bin. What is the expected number of empty bins? What is the expected number of bins with exactly one ball?

i. The indicator variable X_i will be the variable that a given bin is empty after all tosses, it's probability is similar to the procedure done for 5.4.1

$$X_i = P\{empty\} = (\frac{n-1}{n})^n$$

Then the expected value of variable X will be the sum of X_i, which will be the expected number of empty bins.

$$E[X] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} (\frac{n-1}{n})^n = n(\frac{n-1}{n})^n$$

ii. We define another indicator variable Y_i as the probability a given bin contains exactly one ball after all tosses, this is similar to the second part of 5.4.1 as it uses binomial distribution as well, because throws are independent.

$$Y_i = P\{one\ ball\} = \binom{n}{1}(\frac{n-1}{n})^{n-1}(\frac{1}{n})$$

The expected number of bins containing exactly one ball after the tosses are done will be given by the expected value of Y, which gives the number of bins with one ball and one ball only.

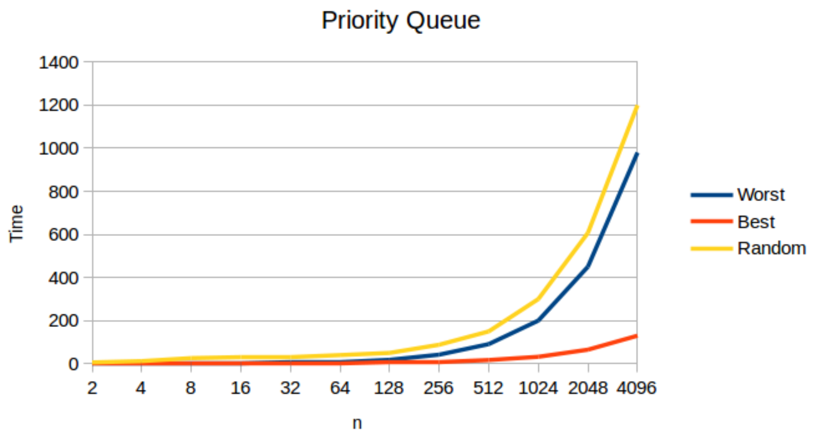
$$E[Y] = \sum_{i=1}^{n} E[Y_i] = \sum_{i=1}^{n} \binom{n}{1}(\frac{n-1}{n})^{n-1}(\frac{1}{n}) = \sum_{i=1}^{n} \frac{n}{n}(\frac{n-1}{n})^{n-1} = n(\frac{n-1}{n})^{n-1}$$

2. Priority Queue

For a Priority Queue, the important methods for this data structure are $Max-Heap-Insert$, $Extract-Max$ and $Heap-Increase-Key$ which according to $Cormen's$: $Introduction\ to\ Algorithms$, have the following complexities:

- $Max-Heap-Insert(A, key)$ has complexity $O(nlgn)$.
- $Extract-Max(A)$ has complexity $O(lgn)$.
- $Heap-Increase-Key(A)$ has complexity $O(lgn)$.

  After testing the code for a Priority Queue in the file *priorityQ.cpp* and registering the results for the best input (elements are inserted in descending order, so less swaps between parent and child occur), worst input (elements are inserted in asceding order, so more swaps happen between parent and child occur), and random input (from a uniform distribution) the following plot of the data resembles the complexity described in the *Cormen*.



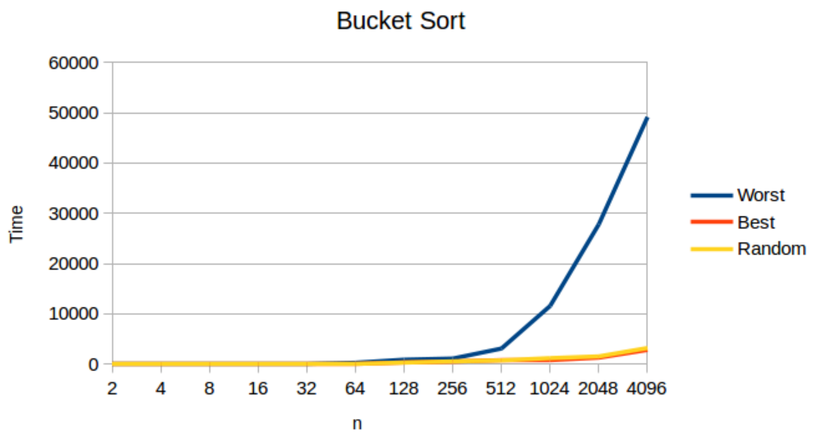
Code for the Priority Queue is included in the included file *priorityQ.cpp*.

3. Linear Sorting

   (a) Bucket Sort
   According to *Cormen's : Introduction to Algorithms*, bucket sort has linear complexity, on average, when the conditions for it are met:

   - Each element in the array satisfies $0 \leq A[i] < 1$.

   After testing the code included in *bucket.cpp* to sort a best input array (elements are equally distributed amongst the buckets), worst input array (elements fall in the same bucket), and random input (elements come from a uniform distribution) the following plot of the data resembles the complexity described in the *Cormen*.

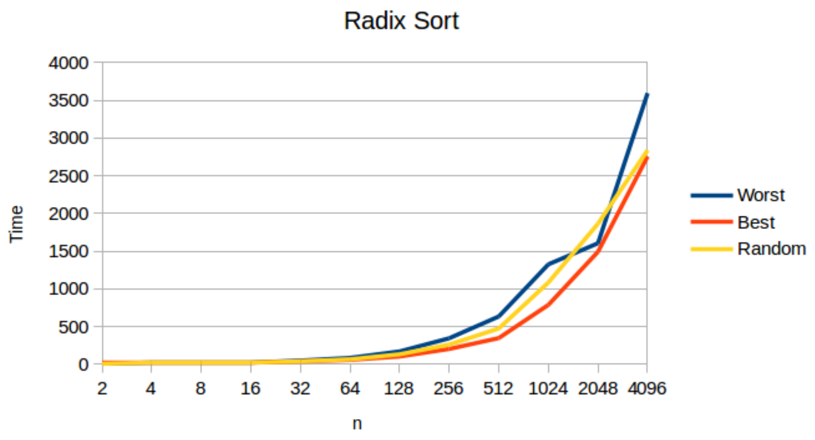


As seen in the graph, the best and random case don't deviate from a linear function $O(n)$.

(b) Radix Sort
According to $Cormen's$: $Introduction\ to\ Algorithms$, radix sort has linear complexity, on average, $O(n)$.

- Radix Sort sorts an array of numbers starting from least to most significant digit, it performs best when the numbers of digits are constant across the array.

After testing the code included in the file *radix.cpp* to sort a best input array (elements in the array have the same number of digits), worst input array (elements in the array vary in digits), and random input array (elements come from a uniform distribution) the following plot of the data resembles the complexity described in the $Cormen$.

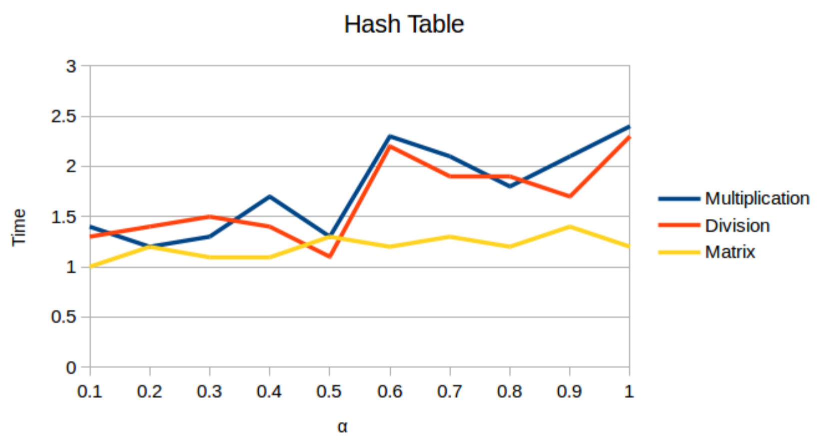


For each n, the time taken closely fits $\frac{1}{2}n$, so the overall complexity can be concluded to be $O(n)$.

4. Hash Table:
Code for the Hash Table implementation using linear lists as a collision policy is included in the file *hashtable.cpp*, where the different hash functions are implemented:

- Division.
- Multiplication.
- Matrix.

Graphing the results of several tests with different alphas, I got almost constant values between 1 and 3, so the complexity described in the $Cormen$ holds for $O(1 + \alpha)$.


Hash Table
Time
α
Multiplication
Division
Matrix
0
0.5
1
1.5
2
2.5
3
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1