Group: 7 due: 20 September 2019

CSCI 432 Problem 2-1

Collaborators: *Peter Gifford, Ren Wall, Kyle Brekke, Madison Hanson*

Give a linear-time algorithm that takes two sorted arrays of real numbers as input, and returns a merged list of sorted numbers. You should give your answer in pseudocode. Your answer should contain:

- A prose explanation of the algorithm.

  This algorithm works by keeping a pointer on one item in each list. Using these pointers we can check how a number in one list compares to a spot in the other list. If the numbers are the same then they can be added both to the list and increment both pointers, if one is smaller than the other then we can add that to the list and increment that pointer. This way the pointers walk through the lists only having to check each item one time and putting them into a new list that is still sorted.

- Psuedocode. (Be sure to review the two resources on pseudocode that were posted as readings for Week 2! I also suggest the algorithm / algorithmx package in LaTex.)

  See Below

- The decrementing function for any loop or recursion.

  Let $\mathbb{X}$ denote the state space of the algorithm. We define the function $D\colon \mathbb{X} \to \mathbb{N} \cup \{0\}$ by $D(\mathbb{X}) = (length(A) + length(B)) - (i + j)$

  Each time through the loop either i, j or both are incremented. The are in $\mathbb{N}$ and therefore eventually $i + j$ must eventually equal $length(A) + length(B)$ and therefore break the loop.

  Let $\mathbb{X}$ denote the state space of the algorithm. We define the function $D\colon \mathbb{X} \to \mathbb{N} \cup \{0\}$ by $D(\mathbb{X}) = length(A) - i$

  Let $\mathbb{X}$ denote the state space of the algorithm. We define the function $D\colon \mathbb{X} \to \mathbb{N} \cup \{0\}$ by $D(\mathbb{X}) = length(B) - j$

  For the two loops above on each iteration, either i or j will be incremented and will eventually hit the same length as the list they are being subtracted from and therefore hit zero and break the loop.

- Justification of why the runtime is linear.

  The algorithm will go through every item in the lists exactly once since the counters i and j will increment until they hit the array size and therefore no item in the lists will have more than O(1) spent on it. All the loops run in O(n) and are not nested with all steps in between taking O(1). This mean that O(A.size()+B.size()) is the complexity and therefore it is run in linear time.

Algorithm 1 Merged list of sorted numbers from two sorted lists

```
1: procedure MERGE(A, B)                ▷ A and B are sorted lists this sorts them into list c
2:     in: Sorted lists A,B
3:     out: Sorted list c, the combination of A and B
4:     c ← list;
5:     i, j ← 0;
6:     while i < A.size()&&j < B.size() do
7:         if A.get(i) == B.get(j) then
8:             c.add(A.get(i), B.get(j));
9:             i++;
10:            j++;
11:        else if A.get(i) < B.get(j) then
12:            c.add(A.get(i));
13:            i++;
14:        else if A.get(i) > B.get(j) then
15:            c.add(B.get(j))
16:            j++;
17:        end if
18:    end while
19:    if i == A.size() then
20:        while j < B.size() do
21:            c.add(B.get(j));
22:            j++;
23:        end while
24:    end if
25:    if j == B.size() then
26:        while i < A.size() do
27:            c.add(A.get(i));
28:            i++;
29:        end while
30:    end if
31:    return c;
32: end procedure
```

Group: 7 due: 20 September 2019

CSCI 432 Problem 2-2

Collaborators: *Peter Gifford, Ren Wall, Kyle Brekke, Madison Hanson*

EPI 15.4 (Generate the Power Set) gives code to compute the power set of a set (without duplicates). Present this problem and solution in your own words using pseudocode.

This algorithm is aimed at getting all the possible sets that can be made from a given set. The solution below uses a form of recursion to repeatedly add and remove elements from the recursion branch so that it can reach the bottom and generate a new set.

Algorithm 2 Power Sets

```
1: procedure GENERATEPOWERSETS(inputSet)                          ▷ Startup function
2:     in: List of integers that is a the inputSet
3:     out: List of Lists of Integers that are the power sets;
4:     powerSet ← list;
5:     newList ← list;
6:     directedPowerSet(inputSet, 0, newList, powerSet);
7:     return powerSet;
8: end procedure
9: procedure DIRECTEDPOWERSET(inputSet, toBeSelected, selectedSoFar, powerSet)
10:    in: inputSet : the original input set, toBeSelected : the spot in inputSet that the algorithm is
       checking, selectedSoFar : list of spots in inputSet already checked, powerSet : list of power sets already
       selected
11:    out: None
12:    if toBeSelected == inputSet.size() then
13:        powerSet.add(selectedSoFar.asList());        ▷ Adds all of selected so far because they represent a
       powerSet to powerSet and ends because there is nothing left to check
14:        return;
15:    end if
16:    selectedSoFar.add(inputSet.get(toBeSelected));
17:    directedPowerSet(inputSet, toBeSelected + 1, selectedSoFar, powerSet);
18:    selectedSoFar.remove(selectedSoFar.size() - 1);
19:    directedPowerSet(inputSet, toBeSelected + 1, selectedSoFar, powerSet);
20: end procedure
```

Group: 7 due: 20 September 2019

CSCI 432 Problem 2-3

Collaborators: *Peter Gifford, Ren Wall, Kyle Brekke, Madison Hanson*

In EPI 15.1 (The Towers of Hanoi Problem), prove that the algorithm as presented terminates. In particular, you should give the decrementing function for the recursion.

Proof:

In order to prove that the provided algorithm terminates, we must show that the recursive component of the algorithm can be represented as a decrementing function.

In order for a given function to be a decrementing function, it must have the following qualities:

1. The function must have an integer value indicating the current conditions of a given loop.
2. The function must have a predefined value greater than zero before entering the loop.
3. The function must have a well-ordered co-domain which it decrements over.
4. The function must terminate when its value reaches zero.

Let D be the function representing the current state of the recursive loop in the provided algorithm. D can be represented as $D(n) = D_1(n-1) + D_2(n-1)$, where $D(n)$ is the value of D at n. This therefor means that $D_1(n-1)$ is the first recursive step in the algorithm, and $D_2(n-1)$ is the second recursive step in the algorithm. Since this function describes an integer value and is applicable to the algorithm, the first requirement holds for D.

D initially begins with a value of $2n$, where n is the number of rings which need to be moved. Because D has a predefined initial value before entering the recursion, the second criterion for D to be a decrementing function holds.

There are two cases which occur during each recursive step of D, we either step into $D_1(n-1)$ or $D_2(n-1)$. In both cases, the value of D decrements by one. This occurs over the co-domain of $(2n, 2n-1, ..., 1, 0)$, which is well-defined with a minimum value 0. Due to this, the third requirement holds for D.

Finally, when D reaches zero, the current recursive loop ends. This holds true for every recursive step of D as well. Because of this, the last requirement holds for D.

Because the aforementioned requirements for D to be a decrementing function have been met, we can conclude that D is a decrementing method which describes the provided algorithm. Because D is a decrementing function, we can conclude that the provided algorithm terminates.

Group: 7 due: 20 September 2019

CSCI 432 Problem 2-4

Collaborators: *Peter Gifford, Ren Wall, Kyle Brekke, Madison Hanson*

For the stock market problem discussed in class on September 6th (and in CLRS 4.1), walk through the algorithm for the following input:

$$\texttt{price} = \{3, 6, 8, 2, 1, 10, 5, 7\}.$$

This algorithm has the goal of taking a list of integers and finding the greatest gap between two numbers where the first number precedes the larger number in the list. We have displayed how this works by showing the inputs to the functions in the book at each step. The input is displayed as $n = array, n[position] : number_a t_p osition$. The entire list is put into the first function. BuySell({n[1]:3, n[2]:6, n[3]:8, n[4]:2{n[5]:1, n[6]:10, n[7]:5, n[8]:7})

The method call above splits the list in half and because the list is longer than two items it calls the BuySell function again with the two halves of the list.
BuySell({n[1]:3, n[2]:6, n[3]:8, n[4]:2}), BuySell({n[5]:1, n[6]:10, n[7]:5, n[8]:7})

The lists have been split in half again and once again get passed into BuySell. This step returns those sets up to the last recursion level to be used by compare in the next step
BuySell({n[1]:3,n[2]: 6}) BuySell({n[3]:8, n[4]:2}), BuySell({n[5]:1, n[6]:10}), BuySell({n[7]:5, n[8]:7})
return = {n[1]:3,n[2]: 6}{n[3]:8, n[4]:2}{n[5]:1, n[6]:10}{n[7]:5, n[8]:7}

Now that the lists are of size 2 or less, the compare function is called on two of the lists as well as the first element of the first list and second element of the second list and returns set of farthest points.
compare({n[1]:3,n[2]:6},{n[3]:2,n[4]:8},{n[1]:3,n[4]:8}) = {n[3]:2,n[4]:8}, compare({n[5]:1,n[6]:10},{n[7]:5, n[8]:7},{n[5]:1,n[8]:7}
= {n[5]:1, n[6]:10}
return = {n[3]:2,n[4]:8}{n[5]:1,n[6]:10}

The same process as the last step is repeated with the new lists generated from the last step.
compare({n[3]:2,n[4]:8},{n[5]:1,n[6]:10},{n[3]:2,n[6]:10}) = {n[5]:1,n[6]:10}

There was only one call of compare and therefore only one set is returned and this set is the answer.
result = {n[5],n[6]}

Group: 7 due: 20 September 2019

CSCI 432 Problem 2-5

Collaborators: *Peter Gifford, Ren Wall, Kyle Brekke, Madison Hanson*

Prove using induction that the closed form of:

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n-1) + n & n > 1 \end{cases}$$

is $O(n^2)$.

Proof by induction:
Base Case: $T(1) = 1$ takes 1 step.
Inductive assumption: $T(n-1)$ takes m steps.
Inductive Step: $T(n)$ takes one more step than $t(n-1)$ so $T(n)$ takes $m+1$ steps.
By induction we can see that $m + 1 = n$ so $T(n)$ takes n steps. $T(n) \in O(n) \subset O(n^2)$

Group: 7 due: 20 September 2019

CSCI 432 Problem 2-6

Collaborators: *Peter Gifford, Ren Wall, Kyle Brekke, Madison Hanson*

What is the closed form of the following recurrence relations? Use Master's theorem to justify your answers:

1. $T(n) = 16T(n/4) + \Theta(n)$

   $a = 16, b = 4, n^2, f(n) = n, case1$
   $\epsilon = 1$ T(n) = $\Theta(n^2)$

2. $T(n) = 2T(n/2) + n \log n$

   $a = 2, b = 2, n^1, f(n) = n \log n, case3$
   $f(n) = \Theta(n^c), c = 2$
   $\log_2 2 < 2$ satisfies condition for case 3
   T(n) = $\Theta(n \log n)$

3. $T(n) = 6T(n/3) + n^2 \log n$

   $a = 6, b = 3, n^1.6, f(n) = n^2, case3$
   $f(n) = \Theta(n^c), c = 2$
   $\log_3 6 < 2$ satisfies condition for case 3
   T(n) = $\Theta(n^2)$

4. $T(n) = 4T(n/2) + n^2$

   $a = 4, b = 2, n^2, f(n) = n^2, case2$
   T(n) = $\Theta(n^2 \log n)$

5. $T(n) = 9T(n/3) + n$

   $a = 9, b = 3, n^2, f(n) = n, case1$
   $\epsilon = 1$ T(n) = $\Theta(n^2)$

Note: we assume that $T(1) = \Theta(1)$ whenever it is not explicitly given.

Group: 7 due: 20 September 2019

CSCI 432 Problem 2-7

Collaborators: *Peter Gifford, Ren Wall, Kyle Brekke, Madison Hanson*

The skyline problem: You are waiting for the ferry across the river to get into a big city, and notice n buildings in front of you. You take a photo, and notice that each building has the silhouette of a rectangle. Suppose you represent each building as a triple (x_1, x_2, y), where the building can be seen from x_1 to x_2 horizontally and has a height of y. Let `rect(b)` be the set of points inside this rectangle (including the boundary). Let `building` be the set of n triples. Design an algorithm that takes `buildings` as input, and returns the skyline, where the skyline is a sequence of (x, y) coordinates defining $\cup_{b \in \texttt{buildings}} \texttt{rect}(b)$.

Goal is to use a divide and conquer algorithm (said in class).

This algorithm takes in the coordinate of a rectangular buildings as x1,y,x2. With these, the outline of the "skyline" they generate is returned using a divide and conquer strategy similar to merge sort. However instead of sorting numbers this returns the better point to be represented on the skyline.

Algorithm 3 Skyline Problem

```
1: procedure getSkyline(buildingCoord)
2:     in: list of skyline variables as buildingCoord.
3:     out: final list of coord for the skyline.
4:     return calculateSkyline(buildingCoord, 0, buildingCoord.size() - 1)
5: end procedure
6: procedure calculateSkyline(arr, l, h)
7:     in: list of coordinate as arr, the low spot in the array as l, the high spot in the array as h
8:     out: list of coordinate for the skyline
9:     if l == h then                          ▷ what method returns once there is no more options to split
10:        res ← list;
11:        res.add(arr[0],arr[1]);
12:        res.add(arr[2],0);
13:        return res;
14:    end if
15:    mid ← (l + h)/2                          ▷ Splitting down the middle like merge sort
16:    listLeft ← calculateSkyline(arr, l, mid);
17:    listRight ← calculateSkyline(arr, mid + 1, h);
18:    toReturn ← mergeSkylines(listLeft, listRight)
19:    return toReturn;
20: end procedure
21: procedure mergeSkylines(left, right)
22:    in: lists of coordinate needed to be merged as left, right
23:    out: merged and 'sorted' lists of the resulting skyline
24:    toReturn ← list
25:    i, j ← 0
26:    heightLeft, heightRight ← 0
27:    while i < left.size()&&j < right.size() do
28:        if left[i][0] < right[j][0] then
29:            x ← left[i][0];
30:            heightLeft ← left[i][2];
31:            maxHeight ← max(heightLeft, heightRight);      ▷ max gets the maximum value of values
   entered since only the tallest building can be seen
32:            toReturn.add((x, maxHeight));
33:            i ← i + 1;
34:        else
35:            x ← right[i][0];
36:            heightRight ← right[i][2];
37:            maxHeight ← max(heightLeft, heightRight);
38:            toReturn.add((x, maxHeight));
39:            j ← j + 1;
40:        end if
41:    end while
42:    while i < left.size() do      ▷ If one list is bigger than the other we need to add all the rest of the
   skylines because we can see all of them.
43:        toReturn ← left[i];
44:        i ← i + 1;
45:    end while
46:    while j < right.size() do
47:        toReturn ← right[i];
48:        j ← j + 1;
49:    end while
50:    return toReturn;
51: end procedure
```

Group: 7 due: 20 September 2019

CSCI 432 Problem 2-8

Collaborators: *Peter Gifford, Ren Wall, Kyle Brekke, Madison Hanson*

The `rand()` function in the standard C library returns a uniformly random number in `[0,RANDMAX-1]`. Does `rand()` mod n generate a number uniformly distributed in $[0, n-1]$?

Note I: This is the second variant in EPI 5.12.

Note II: When asked questions of this form, you are expected to justify your answer.

When you take `rand()` mod n it does generate a number uniformly distributed in $[0, n-1]$. This is because the `rand()` function generates uniformly distributed random numbers, so the numbers generated by `rand()` mod n will be a uniform distribution of numbers with the numeric possibilities of (RANDMAX-1)/(n). This new random distribution would be a uniform distribution in $[0, n-1]$, meaning `rand()` mod n does generate a number uniformly distributed in $[0, n-1]$.

Group: 7 due: 20 September 2019

CSCI 432 Problem 2-9

Collaborators: *Peter Gifford, Ren Wall, Kyle Brekke, Madison Hanson*

Algorithms where we use randomization to find a deterministic answer are known as Las Vegas algorithms. Monte Carlo algorithms also use randomization, but might not always give the right answer; however, they either have a high probability of being correct or close to correct.

(a) Give a Monte Carol algorithm to estimate π.

(algorithm taken from http://www.eveandersson.com/pi/monte-carlo-circle)

Have a circle of radius R inside a square of 2R by 2R.

Generate n numbers randomly.

The numbers that are inside of the circle are M.

$$pi = \frac{4 * M}{N}$$

(b) Let n be the number of random numbers used by your algorithm. Explain why as $n \to \infty$, the expectation of the output for your algorithm is π.

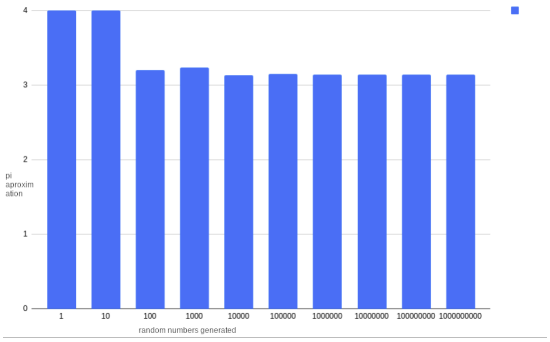
Proof:

As $n \to \infty$, the expectation of our output is π.

Since the area of the square of size 2R by 2R divided by the area of the circle equals to pi/4 the amount of random points that fall in the circle divided by the amount of random points that are generated will be equivalent to that same circle and square ratio, which then divided by four calculates π.

Thus as $n \to \infty$, the expectation of our output is π, which was to be shown.

(c) Implement this algorithm and plot a line graph of the values returned for at least 10 values of n. Code for implementation in Appendix A.



Note: We can use the function `randReal`(a, b) that returns a random real number between a and b inclusive.

Group: 7 due: 20 September 2019

CSCI 432 Problem 2-10

Collaborators: *Peter Gifford, Ren Wall, Kyle Brekke, Madison Hanson*

Appendix A Code for implementation of Monte Carlo Method:

```
import java.util.Random;
public class CarloMonteAndHisPython{
   public static void main(String []args){
      double circleArea = 12.5663706144;
      double squareArea = 16.0;
      double random =0.0;
      double pi = 0.0;
      int inCircle = 0;
      int inSquare = 0;
      int n = 0;
      Random randomizer = new Random();
      for(int i = 0; i ¡1000;i++){
         random =randomizer.nextDouble()*16.0;
         if(random ¿ circleArea){
            inSquare = inSquare + 1;
         }else if(random ¡= circleArea){
            inSquare = inSquare + 1;
            inCircle = inCircle + 1;
         }
         n=i;       }              n=n+1;
      pi = (4.0*(inCircle))/(inSquare);
      System.out.println("Pi using the Monte Carlo method with "+n+" digits is "+pi);
```

```
}
}
```