

CSCI 432 Problem 2-1

Collaborators:

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 $D: \mathbb{X} \rightarrow \cup\{0\}$ 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 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 $D: \mathbb{X} \rightarrow \cup\{0\}$ by $D(\mathbb{X}) = length(A) - i$

Let \mathbb{X} denote the state space of the algorithm. We define the function $D: \mathbb{X} \rightarrow \cup\{0\}$ 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 be have more than $O(1)$ spent on it. 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:    $c \leftarrow \text{list};$ 
5:    $i, j \leftarrow 0;$ 
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
```

CSCI 432 Problem 2-2

Collaborators:

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.

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  $\leftarrow$  list;
5:   newList  $\leftarrow$  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

```

This algorithm is aimed at getting all the possible sets that can be made from a given set. The solution above 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.

Peter Gifford, Ren Wall, Kyle Brekke, Madison Hanson Group: 7

due: 20 September 2019

CSCI 432 Problem 2-3

Collaborators:

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.

CSCI 432 Problem 2-4

Collaborators:

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

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

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})

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})

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}

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}

result = {n[5], n[6]}

Peter Gifford, Ren Wall, Kyle Brekke, Madison Hanson Group: 7

due: 20 September 2019

CSCI 432 Problem 2-5

Collaborators:

Prove using induction that the closed form of:

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

is $O(n^2)$.

CSCI 432 Problem 2-6

Collaborators:

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

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

$$a = 16, b = 4, n^2, f(n) = n, \text{ case 1}$$

$$\epsilon = 1 \quad T(n) = \Theta(n^2)$$

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

$$a = 2, b = 2, n^1, f(n) = n \log n, \text{ case 3}$$

$$f(n) = \Theta(n^c), c = 2$$

$$\log_2 2 < 2 \text{ satisfies condition for case 3}$$

$$T(n) = \Theta(n \log n)$$

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

$$a = 6, b = 3, n^{1.6}, f(n) = n^2, \text{ case 3}$$

$$f(n) = \Theta(n^c), c = 2$$

$$\log_3 6 < 2 \text{ satisfies condition for case 3}$$

$$T(n) = \Theta(n^2)$$

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

$$a = 4, b = 2, n^2, f(n) = n^2, \text{ case 2}$$

$$T(n) = \Theta(n^2 \log n)$$

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

$$a = 9, b = 3, n^2, f(n) = n, \text{ case 1}$$

$$\epsilon = 1 \quad T(n) = \Theta(n^2)$$

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

Peter Gifford, Ren Wall, Kyle Brekke, Madison Hanson Group: 7

due: 20 September 2019

CSCI 432 Problem 2-7

Collaborators:

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 $\mathbf{rect}(\mathbf{b})$ be the set of points inside this rectangle (including the boundary). Let $\mathbf{building}$ be the set of n triples. Design an algorithm that takes $\mathbf{buildings}$ as input, and returns the skyline, where the skyline is a sequence of (x, y) coordinates defining $\cup_{b \in \mathbf{buildings}} \mathbf{rect}(b)$.

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

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
32:       toReturn.add((x, maxHeight));
33:       i ← i + 1;
34:     else
35:       x ← right[j][0];
36:       heightRight ← right[j][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
43:     toReturn ← left[i];                             skylines because we can see all of them.
44:     i ← i + 1;
45:   end while
46:   while j < right.size() do
47:     toReturn ← right[j];
48:     j ← j + 1;
49:   end while
50:   return toReturn;
51: end procedure
```

Peter Gifford, Ren Wall, Kyle Brekke, Madison Hanson Group: 7

due: 20 September 2019

CSCI 432 Problem 2-8

Collaborators:

The `rand()` function in the standard C library returns a uniformly random number in $[0, \text{RANDMAX}-1]$. 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.

CSCI 432 Problem 2-9

Collaborators:

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 Carlo algorithm to estimate π .
- (b) Let n be the number of random numbers used by your algorithm. Explain why as $n \rightarrow \infty$, the expectation of the output for your algorithm is π .
- (c) Implement this algorithm and plot a line graph of the values returned for at least 10 values of n .

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