CS 430 - Recitation Notes

Len Washington III

November 17, 2023

Contents

1.0.1 After Lecture 01 & 02 1
2.0.2 After Lecture 03 & 04 5
3.0.3 After Lecture 05 & 06 8
4.0.4 After Lecture 07 & 08 11
6.0.5 After Lecture 09 & 10 & 11 & 12 13
7.0.6 After Lecture 13 & 14 15
8.0.7 After Lecture 15 & 16 18
9.0.8 After Lecture 17 & 18 19
10.0.9 After Lecture 19 & 20 21
12.0.10 After Lecture 21 & 22 & 23 23
13.0.11 After Lecture 24 & 25 & 26 24

After Lecture 01 & 02 – Answer any questions on HW1
Practice Problems (all taken from previous exams)

1. Which of the following is not true of improved bubble sort (keep track of last swap position on the inner loop and use that to reduce outer loop iterations) on the case on input elements sorted?

   a) It is stable

   b) **Consumes less memory**. Optimized Bubble sort is one of the simplest sorting techniques and perhaps the only advantage it has over other techniques is that it can detect whether the input is already sorted. It is faster than other in case of sorted array and consumes less time to describe whether the input array is sorted or not. It consumes same memory than other sorting techniques. Hence it is not an advantage.

   c) Detects whether the input is already sorted

   d) Consumes less time

2.

Statement 1: In insertion sort, after m passes through the array, the first m elements are in sorted order.

Statement 2: And these elements are the m smallest elements in the array.

   a) Both of the statements are true.

   b) **Statement 1 is true but statement 2 is false**. There may be a smaller value $> m$ indexes from the start of the array, which is why statement 2 is false, however, the first m elements would be sorted amongst themselves.

c) Statement 1 is false but statement 2 is true

d) Both of the statements are false

3. Consider the following program that attempts to locate an element x in a sorted array $a[]$ using binary search. The program is erroneous. Under what conditions does the program fail?

Algorithm 1.1 Erroneous Binary Search

```
function BS
    int i ← 1, j ← 100, k, x            ▷ assume x is assigned a value to search for
    int[] a ← new int[100];             ▷ assume values loaded in sorted order
    repeat
        k ← (i+j)/2
        if a[k] < x then
            i ← k;
        else
            j ← k;
        end if
    until ((a[k] == x) || (i ≥ j))
    if a[k] == x then
        System.out.println("x is in the array")
    else
        System.out.println("x is not in the array")
    end if
end function
```

a) x is the last element of the array $a[]$

b) x is greater than all elements of the array $a[]$

c) **Both of the Above** The above program doesn't work for the cases where element to be searched is the last element of $a[]$ or greater than the last element (or maximum element) in $a[]$. For such cases, program goes in an infinite loop because i is assigned value as k in all iterations, and i never becomes equal to or greater than j. So the while condition never becomes false.

d) x is less than the last element of the array $a[]$

4. What's the worst case of insertion sort if the correct position for inserting element is calculated using binary search?

a) $O(\log n)$

b) $O(n)$

c) $O(n \log n)$

d) $\mathbf{O(n^2)}$ The use of binary search reduces the time of finding the correct position from $O(n)$ to $O(\log n)$. But the worst case of insertion sort remains $O(n^2)$ because of the series of swaps required for each insertion.

5. The following routine takes as input a list of n numbers, and returns the first value of i for which $L[i] < L[i-1]$, or n if no such number exists.

```
int firstDecrease(int* L, int n){
        for(int i=2; i <= n && L[i] >= L[i-1]; i++){}
        return i;
}
```

5a) What is the big-O runtime for the routine, measured as a function of its return value i? $O(n) \to O(i)$. $T(i) = ai + b = O(i)$ (a and b are constants).

5b) If the numbers are chosen independently at random, then the probability that firstDecrease(L) returns i is $\frac{i-1}{i!}$, except for the special case of $i = n+1$ for which the probability is $\frac{1}{n!}$ Use this fact to write an expression for the expected value returned by the algorithm. (Your answer can be expressed as a sum, it does not have to be solved in closed form. Do not use O-notation.) Use expectation

$$\begin{aligned}
\sum_{i=1}^{n+1} \frac{i-1}{i!} &= \frac{1-1}{1!} + \frac{2-1}{2!} + \frac{3-1}{3!} + \cdots + \frac{n-1}{n!} + \frac{(n+1)-1}{(n+1)!} \\
&= \frac{0}{1!} + \frac{1}{2!} + \frac{2}{3!} + \cdots + \frac{n-1}{n!} + \frac{n}{(n+1)!} \\
&= 0 + \frac{1}{2} + \frac{2}{6} + \cdots + \frac{n-1}{n!} + \frac{n}{(n+1)!}
\end{aligned}$$

$$\begin{aligned}
E_{\text{value of } i \text{ returned}} &= \sum_{i=2}^{n+1} [\text{Probability}(\text{return } i) \times i] \\
&= \sum_{i=2}^{n} [\text{Probability}(\text{return } i) \times i] + (\text{Probability}(\text{return } n+1) \times (n+1)) \\
&= \sum_{i=2}^{n} \left[\frac{i-1}{i!} \times i\right] + \frac{1}{n!}(n+1) \\
&= \sum_{i=2}^{n} \left[\frac{i-1}{(i-1)!}\right] + \frac{n+1}{n!} \\
&= \sum_{i=2}^{n} \left[\frac{1}{(i-2)!}\right] + \frac{n+1}{n!}
\end{aligned}$$

5c) What is the big-O average case running time of the routine? Hint: Simplify the previous summation until you see a common taylor series.TODO: Check these corrections

$$\begin{aligned}\sum_{i=2}^{n}\left[\frac{1}{(i-2)!}\right]+\frac{n+1}{n!} &= \frac{1}{(2-2)!}+\frac{1}{(3-2)!}+\frac{1}{(4-2)!}+\frac{1}{(5-2)!}+\cdots+\frac{1}{(n-2)!}+\frac{n+1}{n!}\\ &= \frac{1}{0!}+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{(n-2)!}+\frac{n+1}{n!}\\ &= \frac{1}{1}+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{(n-2)!}+\frac{n+1}{n!}\\ &= 1+\frac{1}{1}+\frac{1}{2}+\frac{1}{6}+\cdots+\frac{1}{(n-2)!}+\frac{n+1}{n!}\\ &= O(e)\end{aligned}$$

6. Some sorting algorithms are NOT stable. However if every key in $A[i]$ is changed to $A[i] * n + i - 1$ (assume $1 \leq i \leq n$) then all the new elements are distinct (and therefore stability is no longer a concern). After sorting, what transformation will restore the keys back to their original values? What is the effect on the runtime of any of the sorting algorithm if you add this transformation before executing the sort and un-transformation after the sort?

$$A[i] \to A[i] * n + i - 1$$

To transform back to the original keys replace each $A[i]$ with $\text{int}[A[i]/n]$, you cannot use $A[i] = (A[i] - i + 1)/n$ because the index i of each value has changed when we sorted. Add $O(2n)$ runtime, which does not affect the runtime of any sort because the growth of those is greater than $O(n)$.

7. a) Use pseudocode to specify a brute-force algorithm that determines when a sequence of n positive integers is given as input, whether there are two distinct terms of the sequence that have as sum a third term. The algorithm should loop through all triples of the sequence, checking whether the sum of the first two terms equals the third.

Algorithm 1.2 Brute Force Sum

function BRUTEFORCESUM(A:int[], n:int)
  for $i \leftarrow 0 \ldots n-1$ **do**
    for $j \leftarrow \underline{i}0 \ldots n-1$ **do**
      for $k \leftarrow \underline{j}0 \ldots n-1$ **do**
        if $\cancel{A[i] \neq A[j]} i \neq j$ && $j \neq k$ && $i \neq k$ && $A[i] + A[j] == A[k]$ **then**
          return true
        end if
      end for
    end for
  end for
  return false
end function

b) Give a big-O estimate for the complexity of the brute-force algorithm from part (a).
$O(n) \times O(n) \times O(n) = O(n^3)$

c) Devise a more efficient algorithm for solving the problem that first sorts the input sequence and then checks for each pair of terms whether their sum is in the sequence.

Algorithm 1.3 Smart Sum

```
function SmartSum(A:int[], n:int)
    ImprovedQuicksort(A)                                  ▷ Runtime: O(n lg n)
    for i ← 0 . . . n − 1 do
        for j ← i . . . n − 1 do
            for k ← j . . . n − 1 do
                if i ≠ j && j ≠ k && i ≠ k && A[i] + A[j] == A[k] then
                    return true
                end if
            end for
        end for
    end for
    return false
end function
```

d) Give a big-O estimate for the complexity of this algorithm. Is it more efficient than the brute-force algorithm?

After Lecture 03 & 04 – Answer any questions on HW1
Practice Problems (all taken from previous exams)

1. Which of the following is time complexity of fun()?

```
int fun(int n){
        int count = 0;
        for(int i = 0; i < n; i++){
                for(int j = i; j > 0; j--){
                        count = count + 1;
                }
        }
        return count;
}
```

a) $O(n)$

b) $O(n^2)$

c) $O(n \log n)$

d) $O(n \log(n) \log(n))$

2. Consider the following function. What is the returned value of the above function?

```
int unknown(int n){
        int i, j, k=0;
        for(i = n/2; i <= n; i++){
                for(j=2; j <= n; j=j*2){
                        k = k + n/2;
                }
        }
        return k;
}
```

a) $\Theta(n^2)$

b) $\mathbf{\Theta(n^2 \log n)}$ The outer loop runs $n/2$ or $\Theta(n)$ times. The inner loop runs $O(\log n)$ times (Note that j is multiplied by 2 in every iteration). So the statement "$k = k + n/2;$" runs $\Theta(n \log n)$ times. The statement increases value of k by $n/2$. So the value of k becomes $n/2 \times \Theta(n \log n)$ which is $\Theta(n^2 \log n)$.

c) $\Theta(n^3)$

d) $\Theta(n^3 \log n)$

3. What is the worst-case auxiliary space complexity (including stack space for recursion) of merge sort?

   a) $O(1)$

   b) $O(\log n)$

   c) $\mathbf{O(n)}$ The worst case is every item is split into its own array at the lowest level of divide.

   d) $O(n \log n)$

4. Choose the incorrect statement about merge sort from the following:

   a) It is a comparison-based sort.

   b) It's runtime is dependent on input order.

   c) It is not an in-place algorithm (all the operations are on the original array).

   d) It is a stable algorithm.

5. Use the definition of big-O to prove or disprove.

   5a) is $2^{n+1}? =?O\left(2^n\right)$ True:

$$2^{n+1} = C2^n$$
$$2 = C \text{ if c } \geq 2$$

   5b) is $2^{2n}? =?O\left(2^n\right)$ False:

$$2^{2n} = C2^n$$
$$2^n = C$$

However 2^n has an infinite range so it cannot be upperbounded

6. Although merge sort runs in $\Theta(n \lg n)$ worst-case time and insertion sort runs in $\Theta(n^2)$ worst-case time, the constant factors in insertion sort make it faster for small n. Thus, it makes sense to use insertion sort within merge sort when sub-problems become sufficiently small. Consider a modification to merge sort in which n/k sub-lists of length k are sorted using insertion sort and then merged using the standard merging mechanism, where k is a value to be determined.

   6a) Show the n/k sub-lists; each of length k, can be sorted by insertion sort in $\Theta(nk)$ worst-case time.

$$k \to \Theta\left(k^2\right)$$
$$\frac{n}{k} \to \Theta\left(\frac{n}{k} \times k^2\right) = \Theta(nk)$$

   6b) Show that the sub-lists can be merged in $\Theta(n \lg\left(\frac{n}{k}\right))$ worst-case time.

$$\frac{n}{k} \to \Theta\left(\frac{n}{2k}\right) \text{ for merging}$$
$$\Theta(2k) \text{ m?? for } 2k \text{ elements}$$
$$\Theta\left(\frac{n}{2k} \times 2k\right) = \Theta\left(n\right) \Theta\left(\frac{n}{4k} \times 4k\right) = \Theta\left(n\right)$$

   6c) Given that the modified algorithm runs in $\Theta(nk + n \lg\left(\frac{n}{k}\right))$ worst-case time, what is the largest asymptotic (Θ-notation) value of k as a function of n for which the modified algorithm has the same asymptotic running time as standard merge sort?

$$nk + n\lg\left(\frac{n}{k}\right) \leq n \lg n$$
$$k + \lg\left(\frac{n}{k}\right) \leq \lg n$$
$$k + \lg(n) - \lg(k) \leq \lg n$$
$$k - \lg(k) \leq 0$$
$$k \leq \lg(k)$$

   6d) How should k be chosen in practice?

7. The Fibonacci sequence 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... is defined recursively as

Algorithm 2.4 The Fibonacci sequence

```
function FIB(n)          ▷ This mathematical definition leads naturally to a recursive algorithm
    if n ≤ 1 then
        return n
    else
        return FIB(n-1) + FIB(n-2)
    end if
end function
```

   7a) Write the recurrence relation, $T(n)$, for the asymptotic runtime for procedure FIB(n) shown above, and solve the recurrence relation to show that $T(n) = O\left(2^{n-2}\right)$.

7b) Another recursive procedure which computes the nth Fibonacci number is below.

Algorithm 2.5

```
function F1(n)
    if n < 2 then
        return n
    else
        return F2(2, n, 1, 1)
    end if
end function

function F2(i, n, x, y)
    if i ≤ n then
        F2(i + 1, n, y, x + y)
    end if
    return x
end function
```

Trace out the algorithm as it computes F1(1), F1(2), F1(3), F1(4), explain how the algorithm works, and then compare its asymptotic runtime to the time for procedure FIB(n).

8. Use mathematical induction to show that when n is an exact power 2, the solution of the recurrence

$$T(n) = \begin{cases} 2 & \text{if } n = 2 \\ 2T(n/2) + n & \text{if } n = 2^k, \text{ for } k > 1 \end{cases}$$

is $T(n) = n \lg n$

After Lecture 05 & 06 – Answer any questions on HW1
Practice Problems (all taken from previous exams)

1. What are the max number of levels in the recursion tree for this recurrence relation?

$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + n$$

   a) $\log_4(n)$
   b) $\log_2(n)$
   c) $\log_{\frac{4}{3}}(n)$
   d) $\log_{\frac{1}{3}}(n)$

2. Under what case of Master's Theorem will the recurrence relation of binary search fail?

   a) 1
   b) 2
   c) 3

d) It cannot be solved using Master's Theorem.

3. What is the purpose of using randomized quick sort over standard quick sort?

   a) Improve the worst-case runtime
   b) To eliminate the possibility that a particular input order will always yied worst-case runtime
   c) To improve accuracy of output
   d) To improve average case time complexity

4. The non-recursive work in quicksort is done in which step of the divide-conquer-combine algorithm?

   a) divide
   b) conquer
   c) combine
   d) none

5. Give big-O bounds of T(n) in each of the following recurrences. Use induction, iteration or Master Theorem.

5a)

$$T(n) = T(n-1) + n;\ T(1) = O(1)$$

$$\begin{aligned}
T(n) &= n + T(n-1) \\
T(n) &= n + n - 1 + T(n-2) \\
T(n) &= n + n - 1 + n - 2 + T(n-3) \\
T(n) &= n + n - 1 + n - 2 + \cdots + T(1) \\
T(n) &= n + n - 1 + n - 2 + \cdots + O(1) \\
T(n) &= \sum_{1}^{n} n \\
T(n) &= O\left(\frac{n^2+n}{2}\right) \\
T(n) &= O(n^2)
\end{aligned}$$

5b)

$$T(n) = 2T\left(\frac{n}{4}\right) + n^{\frac{1}{2}};\ T(1) = O(1)$$

5c)

$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + n^2;\ T(1) = O(1)$$

6. Throughout this course, we assume that parameter passing during procedure calls takes constant time, even if an N-element array is being passed. This assumption is valid in most systems because a pointer to the array is passed, not the array itself. This problem examines the implications of three parameter-passing strategies:

1. An array is passed by pointer. Time $= \theta(1)$.
2. An array is passed by copying. Time $= \theta(N)$, where N is the size of the array.
3. An array is passed by copying only the subrange that might be accessed by the called procedure. Time $= \theta(q - p + 1)$ if the subarray $A[p \dots q]$ is passed. Use $n = q - p + 1$, where n is the size of the subarray passed.

Consider the recursive binary search algorithm for finding a number in a sorted array. Give recurrences for the worst-case running times of binary search when arrays are passed using each of the three methods above, and give good upper bounds on the solutions of the recurrences. Let N be the size of the original problem and n be the size of a subproblem. Binary search works by comparing the element for which you are searching to the elemenet at index $\frac{p-r}{2}$ of a subarray of size n, where p is the first index of the subarray and r is the last index (integer division is used). Therefore, the array passed inot binary search is continually divided in hald.

1)

$$T(n) = T\left(\frac{n}{2}\right) + O(1) \text{ with } T(1) = O(1)$$

The array is passed by pointer, which is constant time. Therefore, the time involved in tha

2)

7. Use the definition of Θ and induction to prove that the recurrence $T(n) = T(N-1) + \theta(n)$ (worst case Quicksort) has the solution $T(n) = \Theta(n^2)$. Since we are not given any boundary conditions, we cann assume the basis step for the inductive proof. Assume the claim is true for $n = k$.
$T(k) = \theta(k^2)$, in other words assume $c_1 k^2 \leq T(k) \leq c_2 k^2$ for some $c_1 > 0$, $c_2 > 0$ and k large enough.
Use that to prove the claim

8. What is you are sorting a collection of data that can have multiple entries of some of the values. When calling Quicksort's PARTITION(A, p, r), where do elements equal to the pivot end up and why?
How could we modify Quicksort and Partition (write pseudocode) so that if we happen to partition on a pivot that had many duplicate values, we can improve the runtime of Quicksort by having smaller recursive calls?

Algorithm 3.6 Quicksort1

function QUICKSORT1(A, p, r) ▷
  if $p < r$ **then**
    $(q_1, q_2) =$ PARITION1(A, p, r) ▷ two return values
    $(()A, p, q_1 - 1)$
  end if
end function

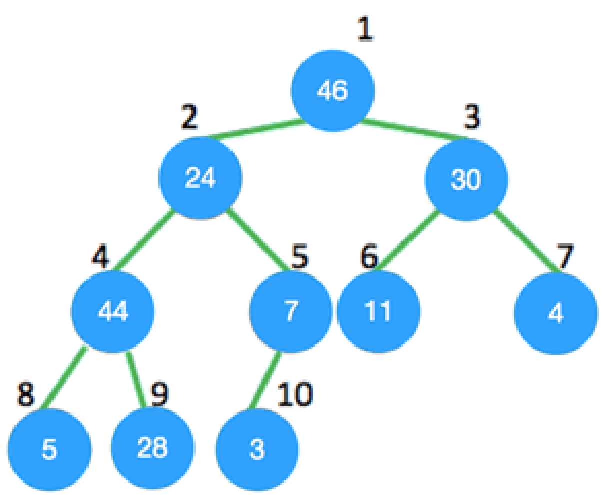
Algorithm 3.7 Partition1

```
function PARTITION1(A, p, r)                ▷ Two return values
    endLow ← p − 1, endEqual ← p − 1
    pivot ← A[r]
    for j=p to r-1 do
        get function
    end for
end function
```

After Lecture 07 & 08 – Answer any questions on HW2 (due today)
Practice Problems (all taken from previous exams)

1. Which one of the following is false?
   a) Heap sort is an in-place algorithm.
   b) Heap sort has $O(n \log n)$ average case time complexity.
   c) Heap sort is a stable sort.Heap sort is an in-place algorithm as it needs $O(1)$ auxiliary space.
   d) Heap sort is a comparison-based sorting algorithm.

2. Consider the max heap shown below, the node with value 24 violates the max-heap property. Once heapify procedure is applied to it, which position will it be in?



   a) 5
   b) 8
   c) 9
   d) You cannot call heapify at the node with value 24

3. Counting sort can be used on any numeric data.

a) TRUE, it can be used, but it is a very bad idea. Counting sort requires a very large range of numbers, counting sort requires a very large array. This reduces its memory efficiency and increases space consumption. So while it possible to be used, it isn't a good idea to do so on any numeric data.

b) FALSE

4. Which of the following is not true about all comparison based sorting algorithms?

   a) The minimum possible runtime growth on a random input is $O(n \log n)$.

   b) Can be made stable by also using position when two elements are compared.

   c) Counting Sort is not a comparison-based sorting algorithm.

   d) Merge Sort is a comparison-based sorting algorithm.

5. The BUILD-MAX-HEAP discussed in class and shown to be $O(n)$ uses this process. Call Heapify from heap index position $\lfloor \frac{heapsize}{2} \rfloor$ down to heap index position 1. Building a heap can also be implemented by starting with an empty heap and repeatedly using MAX-HEAP-INSERT to insert the elements into the heap. Consider the following implementation:

```
function BUILD-MAX-HEAP1(A)
    H ← empty heap (of max size A.length)
    for i = 1 to A.length do
        H.MAX-HEAP-INSERT(A[i])
    end for
end function
```

   a) Do the procedures BUILD-MAX-HEAP and BUILD-MAX-HEAP1 always create the same heap when run on the same input array? Prove that they do, or provide a counterexample. The procedures do not always create the same heap when run on the same input array.

   b) Show that in the worst case, BUILD-MAX-HEAP1 requires $\Theta(n \lg n)$ time to build an n-element heap. Since all but the last level is always filled, the height h of an n element heap is boundend because $\sum_{i=0}^{h} 2^i = 2^{h+1} - 1 = \ldots$

6. The operation HEAP-DELETE(A, i) deletes the item in node i from heap A. Give an implementation of HEAP-DELETE that runs in $O(\lg n)$ time for an n-element max-heap.

Algorithm 4.8 HEAP-DELETE

```
function HEAPDELETE(A, i)
    swap A[i] with A[heapsize]    ▷ move the item to be deleted to the last position in the heap
    decrement heapsize
    key ← A[i]
    if key ≤ A[PARENT(i)] then
        HEAPIFY(A, i)
    end if
    ...
end function
```

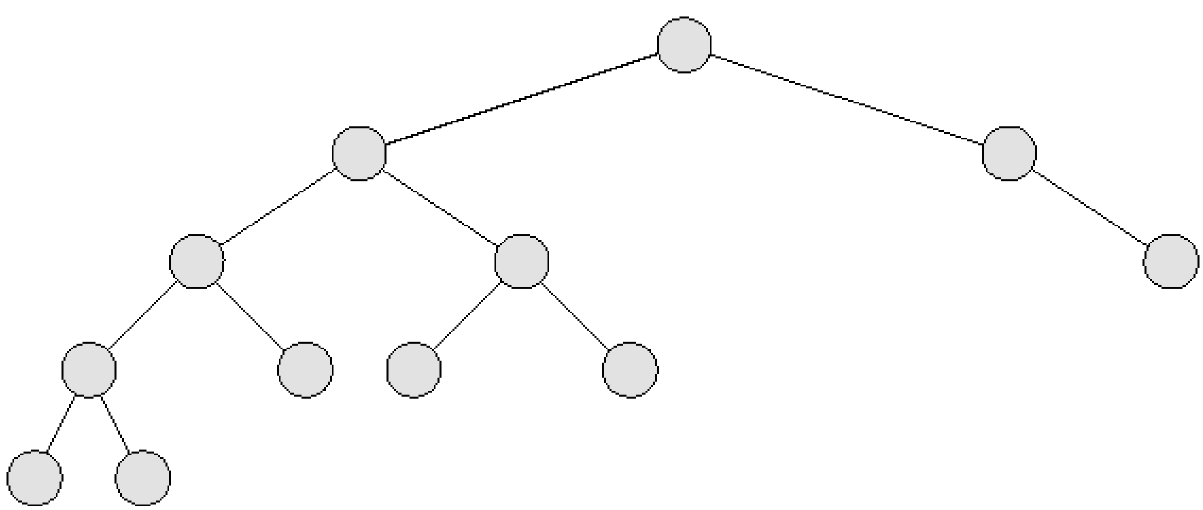
7. Professor Fermat has the policy of giving A's to the top $n^{\frac{1}{2}} = \sqrt{n}$ students of his class, where n is the number of students. The algorithm that he uses to determine the top $n^{\frac{1}{2}}$ students first sorts the list of students by their numerical, real-valued grade, and then picks the top $n^{\frac{1}{2}}$ students from the sorted list. This algorithm has time-complexity $O(n \log n)$ because of the sort. Can you suggest a more efficient algorithm that has time-complexity $O(n)$? Describe your algorithm informally in English and justify its time-complexity. You can use a MaxHeap to effectively sort the students based off of the students grades which takes $O(n)$. Now perform RemoveMax $\sqrt{n}$ times. Each RemoveMax operation takes time $O(\log N)$. Total time complexity of this algorithm is $O(n) + \log(n)\sqrt{n} = O(n)$ because $\log(n)\sqrt{n} = O(n)$, because $\log(n) = O\left(\sqrt{n}\right)$

After Lecture 09 & 10 & 11 & 12 – Answer any questions on HW3 (due today)
Practice Problems (all taken from previous exams)

1. You should repeatedly use the $O(n)$ median algorithm to find the best choice for the order to insert values in a binary search tree to ensure that the binary search tree is balanced.
   a) True.
   b) False.
2. If you were just given the output of a traversal of a valid binary tree, could you draw the actual BST? Which of the following traversals is/are needed to draw the BST from given traversals.

   1) Inorder 2) Preorder 3) Postorder

   a) Any one of the given three traversals is sufficient.
   b) Either 2 or 3 is sufficient.
   c) Both 2 and 3 are needed.
   d) Both 1 and 3 are needed.
3. The nodes of the following tree can be colored such that it is a red-black tree.



a) True.

b) False.

4. Insert 20, 15, 5, in that order, into an empty binary search tree. What rotations would be needed to balance the tree?

   a) Left rotation about node (5).

   b) Right rotation about node (20).

   c) Left rotation about node (15), followed by a right rotation about node (20).

   d) No rotations are needed.

5. Assume that you are given a “black-box” (i.e., you do not have the source code) procedure MEDIAN that takes as parameters an array A and subarray indices p and r, and returns the value of the median element of $A[p \dots r]$ in $O(n)$ time in the worst case. Give a simple, recursive, linear-time algorithm that uses this procedure MEDIAN to find the ith smallest element. Write the recurrence relation for your algorithm and show the solution is linear growth.

Algorithm 6.9

```
function SELECT(A, p, r, i)
    if p == r then
        return A[p];
    end if
end function
```

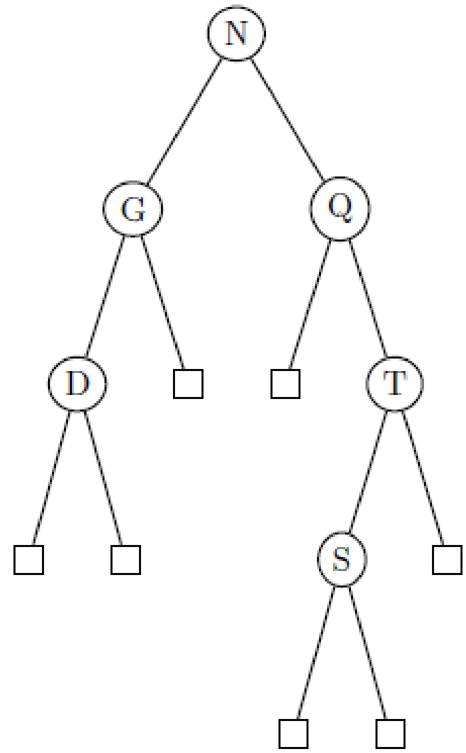
6. BST Given an arbitrary binary tree T with integer keys stored at the nodes and pointers to the left and right sub binary trees, design an efficient algorithm which determines whether or not T is a binary search tree. What is the time complexity of your algorithm? We may determine whether or not a given binary tree T is a binary search

7.

   7a) Show each red-black tree that results after successively inserting the keys 4 7 12 15 3 5 14 18 into an initially empty red-black tree. At the steps where a reb-black tree rule is violated, explain how it is corrected.

   7b) Now delete these keys in this order and show each resultant red-black tree 18 15 7 14. At the steps where a red-black tree rule is violated, explain how it is corrected.

8. The external path length of a binary tree is the sum, taken over all nil leaves of the tree, of the depth of each leaf. Given the following binary search tree, in which internal nodes are shown as circles and external nodes (leaves; nil pointers) are shown as small boxes. The external path length (for nil leaves left to right) is $\mathbf{3+3+2+2+4+4+3=21}$.



8a) Is there a binary search tree on the same set of letters, $\{D, G, N, Q, S, T\}$, that has lower external path length? Either give such a tree and how it is formed from the original BST or prove it does not exist. ~~Yes, rotate the BST such that Q is now the root.~~ Yes, we can reorganize the tree (rotate right at T, then rotate left at Q)

8b) Can the nodes in the original BST be colored red and black to form a proper red-black tree? Yes.No, the root must be black. Then G must be black (if it were red, D would have to be black and blackheight from the root of the nil leaf at the right of G would be 1, but the blackheight from the root to the nil leaves below D would be 2) and D must be red–this gives blackheight of 2 from the root to the nil leaves in the left subtree of N. In the right subtree of the root, the balckheight from N to the nil lead to the left of Q has to be 2 also, so Q must be black. But at least T and S must be black (because we cannot have two red nodes in a row), meaning that the blackheight from N to nil leaves below S would be 3.

8c) What might you conclude about a relationship between external path length and red-black trees? A BST that does not have its minimum possible external path length cannot be a red-black tree.

After Lecture 13 & 14 Practice Problems (all taken from previous exams)

1. If you want to create in order-statistic tree (which needs the size of each subtree rooted at each node), from an already created red-black tree, you can:

   a) perform a pre-order traversal of the order-statistic tree and sum the sizes of each subtree of a node and add one to get the size of each node (nodes with no children assigned size=1)

   b) perform an in-order traversal of the order-statistic tree and sum the sizes of each subtree of a node and add one to get the size of each node (nodes with no children assigned size=1)

c) perform a post-order traversal of the order-statistic tree and sum the sizes of each subtree of a node and add one to get the size of each node (nodes with no children assigned size=1)

2. How does an augmented data structure differ from a traditional data structure?

   a) Augmented data structures have an asymptotically higher memory overhead.
   b) Augmented data structures worsen the asymptotic runtime of basic operations.
   c) Augmented data structures offer additional operations or information.
   d) Augmented data structures have a faster runtime complexity than the non-augmented data structure.

3. If a problem can be broken into sub-problems which are reused several times, the problem has ____.

   a) Overlapping subproblems
   b) Optimal substructure
   c) Memoization[1]
   d) Greedy

4. What is the space complexity of the dynamic programming implementation of the matrix chain problem?

   a) $O(1)$
   b) $O(n)$
   c) $O(n^2)$
   d) $O(n^3)$

5. Given an element x in an n-node order statistic tree and a natural number i, how can we determine the ith successor of x in the linear order of the tree in $O(\lg n)$ time? So x is a key in the tree and we want to find the ith key after x in linear order.

[1] **Memoization** means that we should never try to compute the solution to the

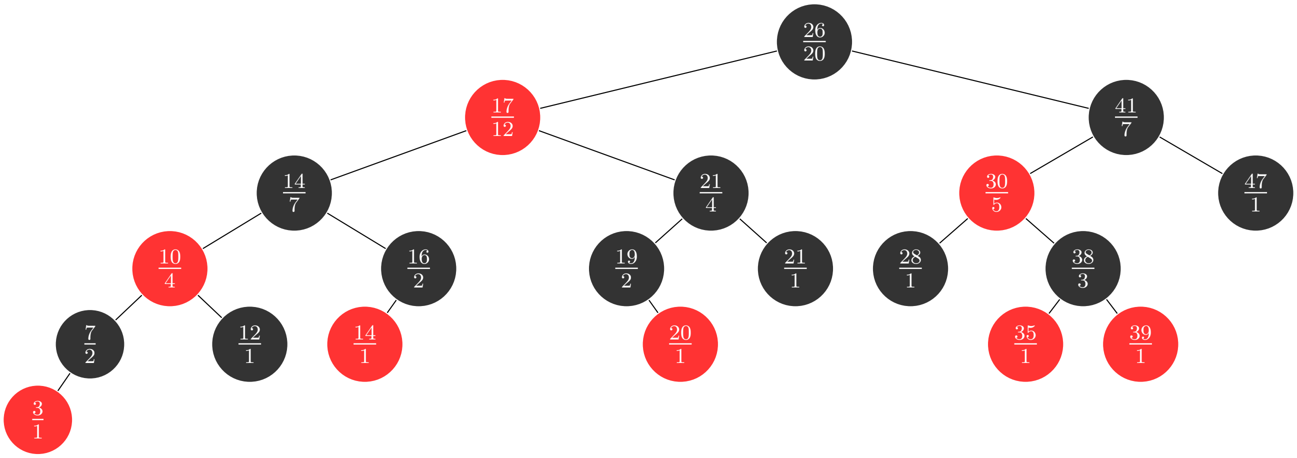


Figure 7.1: An order-statistic tree, which is an augmented red-black tree. In addition to its usual attributes, each node x has an attribute $x.size$, which is the number of nodes, other than the sentinel, in the subtree rooted at x.

First we determine the rank of x by calling OS-RANK(T, x) and name this number r. Then the ith successor of x is actually an element in the tree with rank $r + i$. Hence we call OS-SELECT($T.root$, $r + i$). Both these calls require $O(\lg n)$ time which in total is again $O(\lg n)$.

6. Suppose that the dimensions of the matrices A, B, C, and D are 8×5, 5×11, 11×6, and 6×9 respectively, and that we want to parenthesize the product $ABCD$ in a way that minimizes the number of scalar multiplications. Find the m and s tables computed by MATRIX-CHAIN-ORDER to solve this problem and show the optimal parenthesization.

Table 7.1:

m	A	B	C	D
A	0	440	570	960
B		0	330	600
B			0	594
B				0

7. Let $R(i, j)$ be the number of times that table entry $m[i, j]$ is referenced while computing other table entries in a call of MATRIX-CHAIN-ORDER. Show that the total number of references for the entire table is

$$\sum_{i=1}^{n} \sum_{j=i}^{n} R(i, j) = \frac{n^3 - n}{3}$$

$$\begin{aligned}\sum_{i=1}^{n}\sum_{j=i}^{n} R(i,j) &= \sum_{l=2}^{n}\sum_{i=1}^{n-l+1}\sum_{k=i}^{i+l-2} 2 \\ &= \sum_{l=2}^{n}\sum_{i=1}^{n-l+1} 2(l-1) \\ &= \sum_{l=2}^{n} 2(l-1)(n-l+1) \\ &= \sum_{l=1}^{n-1} 2l(n-1)\end{aligned}$$

After Lecture 15 & 16 – Answer any questions on HW4 (due today)
Practice Problems (all taken from previous exams)

1. In dynamic programming, the technique of storing the previously calculated values is called ____________
   a) Saving value property
   b) Storing value property
   c) Memoization
   d) Mapping
2. What is the time complexity of the brute force algorithm used to find the longest common subsequence for sequence length m and sequence length n $(m < n)$?
   a) $O(mn)$
   b) $O((mn)^2)$
   c) $O(n2^m)$
   d) $O(2^m 2^n)$
3. When dynamic programming is used, it takes less time compared to algorithmic methods that don't utilize overlapping subproblems.
   a) True.
   b) False.
4. Using the dynamic programming solution, determine an LCS of $\{1,0,0,1,0,1,0,1\}$ and $\{0,1,0,1,1,0,1,1,0\}$. Show all your work.
5. Given a sequence of n numbers a_1, a_2, a_3, ..., a_n (some of them might be negative) stored in an array, we want to find two indicies $i \le j$ such that the sum of the numbers from a_i to a_j is maximum, among all possible i j pairs $1 \le i \le j \le n$.

5a) Write pseudocode to sum each contiguous subsequence (from a_i to a_j) and keep track of the maximum one. What is the runtime of your algorithm? The runtime is $O(n^2)$

Algorithm 8.10 Maximum Subsequence

```
function MaxSubsequence
    bestval ← −∞
    for i ← 1...n do
        sumCurrent ← a[i]
        if sumCurrent > bestval then
            bestval ← sumCurrent
            besti ← i
            bestj ← i
        end if
        for j ← i + 1...n do
            sumCurrent ← sumCurrent + a[j]
            if sumCurrent > bestval then
                bestval ← sumCurrent
                besti ← i
                bestj ← j
            end if
        end for
    end for
    return bestval, besti, bestj
end function
```

5b) Now find an $O(n)$ algorithm. Give pseudocode.

Algorithm 8.11 Improved Maximum Subsequence

```
function ImprovedMaximumSubsequence
    M[j] ← max sum over all contiguous sequences ending at a[j]
    a[j] ← either extends the previous contiguous sequence, or a[j] starts a new contiguous
  sequence
    M[j] ← max{M[j − 1] + a[j], a[j]}
end function
```

6. Prove that a binary tree that is not full (every node has 0 or 2 children) cannot correspond to an optimal prefix code. An optimal prefix code is a predix code that gives the shortest possible encoded file length. If we have a prefix code that corresponds to a binary tree that is not full, let n be a node that only has 1 child. Then we could form another binary tree by removing n and moving up n's child. The codewords of all the characters that were descendants of n have now all be decreased by 1, and so the original binary tree could not correspond to an optimal prefix code.

After Lecture 17 & 18 Practice Problems (all taken from previous exams)

1. Which of the following problems is equivalent to the 0–1 Knapsack problem?

a) You are given a bag that can carry a maximum weight of W. You are given N items which have a weight of $\{w_1, w_2, w_3, \ldots, w_n\}$ and a value of $\{v_1, v_2, v_3, \ldots, v_n\}$. You can break the items into smaller pieces. Choose the items in such a way that you get the maximum value.

b) You are studying for an exam and you have to study N questions. The questions take $\{t_1, t_2, t_3, \ldots, t_n\}$ time(in hours) and carry $\{m_1, m_2, m_3, \ldots, m_n\}$ marks. You can study for a maximum of T hours. You can either study a question or leave it. Choose the questions in such a way that your score is maximized.

c) You are given infinite coins of denominations $\{v_1, v_2, v_3, \ldots, v_n\}$ and a sum S. You have to find the minimum number of coins required to get the sum S.

d) You are given a suitcase that can carry a maximum weight of 15kg. You are given 4 items which have a weight of $\{10, 20, 15, 40\}$ and a value of $\{1, 2, 3, 4\}$. You can break the items into smaller pieces. Choose the items in such a way that you get the maximum value.

2. A greedy algorithm can be used to solve all the dynamic programming problems.

   a) True.

   b) False.

3. All optimization problems exhibit optimal substructure.

   a) True.

   b) False.

4. Given a value N, if we want to make change for N cents, and we have an infinite supply of each of $S = \{S_1, S_2, \ldots, S_m\}$ valued coins, how many ways can we make the change? The order of coins doesn't matter, so different permutations of the same coin sets are ignored. Prove the necessary traits of the poblem to determine an algorithm.

   For example, for $N = 4$ and $S = \{1, 2, 3\}$, there are four solutions $\{1, 1, 1, 1\}$, $\{1, 1, 2\}$, $\{2, 2\}$, $\{1, 3\}$. So output should be 4. So the output should be 4. For $N = 10$ and $S = \{2, 5, 3, 6\}$, there are five solutions: $\{2, 2, 2, 2, 2\}$, $\{2, 2, 3, 3\}$, $\{2, 2, 6\}$, $\{2, 3, 5\}$, and $\{5, 5\}$. So the output should be 5. **Optimal Substructure** To count total number solutions, we can divide all set solutions in to two sets.

   1) Solutions that do not contain mth coin (of S_m).

   2) Solutions that contain at least one S_m.

   Let $\text{COUNT}(S[], m, N) = \text{COUNT}(S[], m-1, N) + \text{COUNT}(S[], m, N - S_m)$.
   The problem has optimal substructure property as the problem can be solved using optimal solutions to subproblems. The order of the coins in $S[]$ is not relevant.

   Overlapping Subproblems It should be notes that the above function computes the same subproblems again and again. See the following recursion tree $S = \{1, 2, 3\}$ and $n = 5$. The function $\text{COUNT}(\{1\}, 3)$ is called two times. If we draw the complete tree, then we can see that there are many subproblems being called more than once.

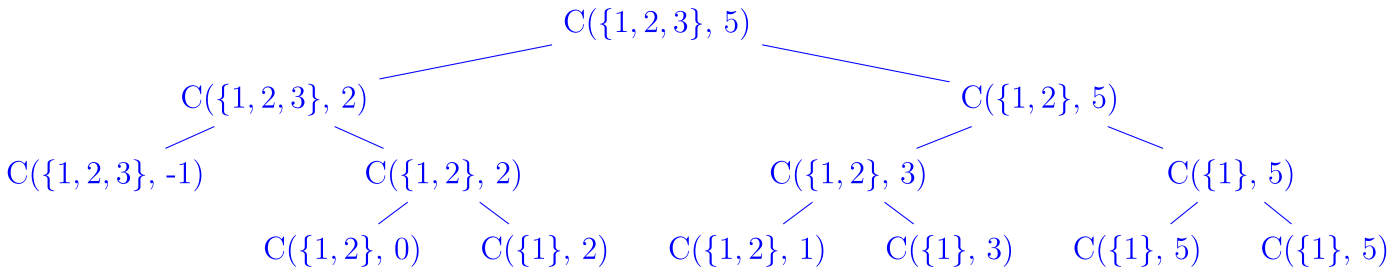


Figure 9.2:

5. Suppose you are managing the construction of billboards on a highway of length M miles. The possible sites for billboards are given by increasing mile numbers $x_1, x_2, \ldots, x_n$ each in the interval $[0, M]$. If you place a billboard at location x_i, you receive a revenue of $r_i > 0$.

   Regulations require that no two billboards be within less than or equal to 5 miles to each other. You'd like to place billboards at a subset of the sites so as to maximize your revenue, subject to this restriction.

   For example, suppose $M = 20$, $n = 4$, $\{x_1, x_2, x_3, x_4\} = \{6, 7, 12, 14\}$, $\{r_1, r_2, r_3, r_4\} = \{5, 6, 5, 1\}$. Then the optimal solution would be to place billboards at x_1 and x_3 for a total revenue of 10.

   Give an efficient algorithm that takes an instance of this problem as input and returns:

   a) the maximum total revenue that can be obtained from any valid subset of sites, and
   b) a subset of sites that achieves this maximum total revenue.

6. Suppose that we have a set of n activities, each with start time s_i and finish time f_i, to schedule among a large number of lecture halls. We wish to schedule all the activities using as few lecture halls as possible. Give an efficient algorithm to determine which activity should use which lecture hall. Prove that your algorithm is optimal and discuss the asymptotic runtime of your algorithm. DO NOT JUST USE THE GREEDY-ALGORITHM ONCE PER ROOM; there is a more efficient algorithm.

After Lecture 19 & 20 – Answer any questions on HW5 (due today)
Practice Problems (all taken from previous exams)

1. The step driving the runtime in the solution to the fractional knapsack problem is ________.

   a) Breaking items into fractions.
   b) Adding items into the knapsack.
   c) Sorting.
   d) Looping through sorted items.

2. How many bits are needed for fixed length encoding if the size of the character set is X?

   a) $\lg X$

b) $X + 1$

c) $2X$

d) X^2

3. The characters a to h have the set of frequencies based on the first 8 Fibonacci numbers as follows: a: 1, b: 1, c: 2, d: 3, e: 5, f: 8, g: 13, h: 21
A Huffman code is used to represent the characters. What is the sequence of characters corresponding to the following code? 110111100111010

a) $fdheg$

b) $ecgdf$

c) $dchfg$

d) $fehdg$

4. If a data structure supports an operation FOO such that a sequence of n FOO's takes $O(n \lg n)$ time in the worst case, then the amortized time of a foo operation is (answer A) while the actual time of a single FOO operation could be as low as (answer B) and as high as (answer C).

a) $A = n$ $\quad B = \lg n$ $\quad C = n$

b) $A = \log n$ $\quad B = 1$ $\quad C = n \log n$

c) $A = \log n$ $\quad B = 1$ $\quad C = n^2$

d) $A = n$ $\quad B = 1$ $\quad C = n \log n$

5. A pharmacist has W pills and n empty bottles. Let $\{p_1, p_2, p_3, \dots, p_n\}$ denote the number of pills that each bottle can hold. Describe a greedy algorithm, which, given W and $\{p_1, p_2, p_3, \dots, p_n\}$, determine the fewest number of bottles needed to store pills. Prove that your algorithm is correct.

6. How would you modify your pills to bottles algorithm if each bottle also has an associated cost c_i, and you want to minimize the total cost used to store all the pills. Give a recursive formulation of the problem. We want to find the minimum cost obtainable when storing j pills using bottles chosen from the set bottle 1 through bottle i. This occurs either with or without bottle i.

7. You are to maintain a collection of items and support the following operations.

(i) INSERT(item, list): insert item into list (cost = 1)

(ii) SUM(list): sum the items in list, and replace the list with a list containing one items that is the sum (cost = length of list)

Use the Accounting Method to show that the amortized cost of an insert operation is $O(1)$ and the amortized cost of a sum operation is $O(1)$. We will maintain the invariant that every item has one credit. INSERT gets 2 credits, which covers one for the actual cost and one to satisfy the invariant. SUM gets one credit, because the actual cost of summing is covered by the credits in the list, but then the result of the sum will need one credit to maintain the invariant. A common error was not putting a credit on the newly created sum.

After Lecture 21 & 22 & 23
Practice Problems (all taken from previous exams)

1. The number of trees in a binomial heap with n nodes is
   a) $\log n$
   b) n
   c) $n \log n$
   d) $\frac{n}{2}$
2. Which two Fibonacci heap functions have the same complexity?
   a) Insertion, Union
   b) Insertion, Deletion
   c) ExtractMin, Insertion
   d) Union, Delete
3. If $|V|$ is the total number of elements, in the worst case, how many leader pointer updates are needed when fusing two groups in the union method:
   a) $O(1)$
   b) $O(\log |V|)$
   c) $O(|V|)$
   d) $O(|V|^2)$
4. Consider the following program:

```
for i in range(1, 13): # 1 to 12
        MakeSet(i)
Union(2,10)
Union(7,5)
Union(6,1)
Union(3,4)
Union(5,11)
Union(7,8)
Union(7,3)
Union(2,12)
Union(9,6)
Print(Find(6))
Print(Find(3))
Print(Find(11))
Print(Find(9))
```

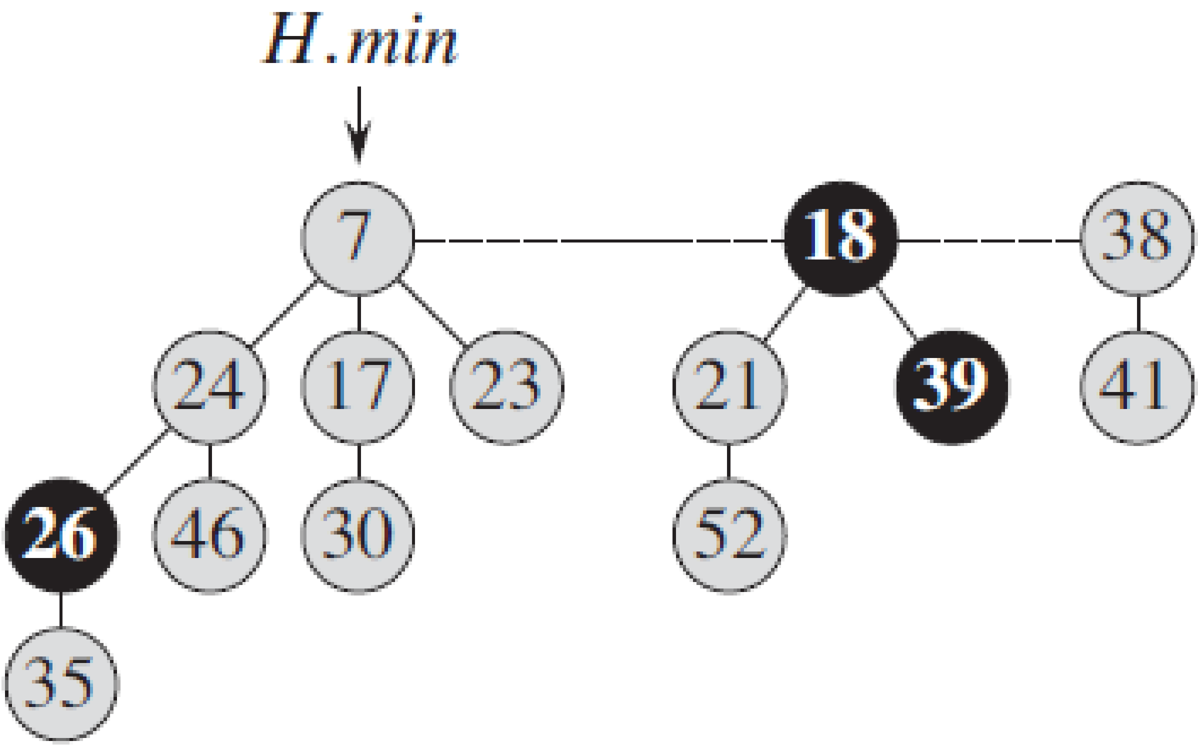
   Assume the disjoint set data structure is implemented so after a union, the smallest valued element in the set is the label of the set. What is the output?
   a) 6 3 11 9

b) 3 1 1 3

c) 1 3 3 1

d) 9 11 11 9

5. Show the Fibonacci heap that results from calling FIB-HEAP-EXTRACT-MIN on the Fibonacci heap shown



Remove 7 from the list and add its children to the root list. Then remerge.

6. We have students $1, 2, \ldots, n$ who need to be assigned to dormitories at a university that has an arbitrarily large number of dorms. There are m same dormitory requests $(s_1, t_1), (s_2, t_2), \ldots, (s_m, t_m)$ meaning students s_i and t_i must be assigned to the same dorm. There are also k different dormitory requests $(u_1, v_1), (u_2, v_2), \ldots, (u_k, v_k)$ meaning students u_i and t_i must be assigned to different dorms. Give an algorithm using the UNION-FIND structure to determine whether it is possible to assign students to forms so that all constraints are satisfied.

After Lecture 24 & 25 & 26 – Answer any questions on HW6 (due today)
Practice Problems (all taken from previous exams)

1. The edges used in a Depth First Search traversal of a graph will result in?

a) Linked List

b) Tree

c) Graph with back edges

d) Min Heap

2. In Depth First Search, how many times a node is checked to see if it has been visited yet?

a) Once

b) Twice

c) Equivalent to number of indegree of the node

d) Thrice

3. Regarding implementation of Breadth First Search using queues, what is the maximum difference in the distance from the source for two nodes present in the queue? (considering each edge length 1)

a) Can be anything

b) 0

c) At most 1[2]

d) Insufficient information

4. Topological sort can be applied to which of the following graphs?

a) Undirected Cyclic Graphs

b) Directed Cyclic Graphs

c) Undirected Acyclic Graphs

d) Directed Acyclic Graphs

5. Which of the following is false?

a) The spanning trees do not have any cycles

b) MST have $n - 1$ edges if the graph has n edges

c) Edge e belonging to a cut of the graph (partitions the vertices of a graph into two disjoint subsets), if has the weight smaller than any other edge in the same cut, then the edge e is present in all the MSTs of the graph.

d) Removing one edge from the spanning tree will not make the graph disconnected

6. Which of the following is false about the Kruskal's algorithm?

a) It is a greedy algorithm

b) It constructs MST by selecting edges in increasing order of their weights

c) It can accept cycles in the MST

d) It uses disjoint set data structure

7. Kruskal's algorithm (pick minimum edge in the graph between two vertices that are not yet in the same connected component) is best suited for the dense graphs than the Prim's algorithm (pick minimum edge from visited to unvisited vertex).

[2] Because the source is moving

a) True

b) False

8. Reword the statement below as a theorem about graphs and then prove it. Assume that is A is a friend of B, then B is a friend of A and that for all A, A is not a friend of A.

   - In any group of $n \geq 2$ people, there are two people with the same number of friends in the group.

   Worded

9. Argue that in a breadth-first search, the value $d[u]$ assigned to a vertex u is independent of the order in which the vertices in each adjacency list are given. Using the graph shown as an example, show that the breadth-first tree computed by BFS can depend on the ordering within adjacency lists.

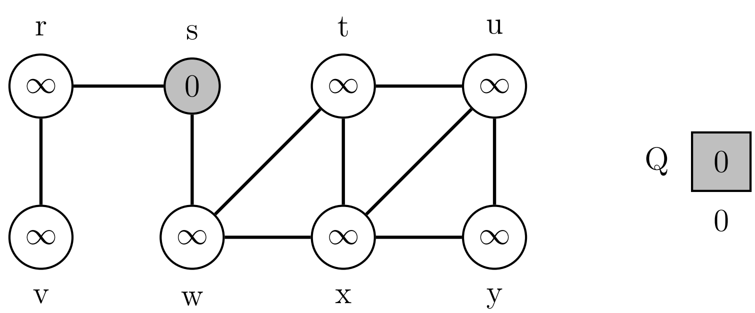


Figure 13.3:

10. 10a. Show how on a directed graph that depth first search starting at vertex u can result in vertex v not being reachable from u even though both u and v have incoming and outgoing edges.

    10b. If a directed graph contains a path from u to v, show that it is not necessary that the $\texttt{S}(v) < \texttt{F}(u)$. $\texttt{S}(\ )$ is the depth first search start time and $\texttt{F}(\ )$ is the depth first search finish time. Consider the directed graph below. If we run the DFS on this directed graph from w, then we can't visit

11. Bob loves foreign languages and wants to plan his course schedule to take the following nine language courses: LA15, LA16, LA22, LA31, LA32, LA126, LA127, LA141, and LA169. The course prerequisites are:

LA15: (none)

LA16: LA15 is prerequisite

LA22: (none)

LA31: LA15 is prerequisite

LA32: LA16 and LA31 is prerequisite

LA126: LA22 and LA32 is prerequisite

LA127: LA16 is prerequisite

LA141: LA16 and LA22 is prerequisite

LA169: LA32 is prerequisite

Find a sequence of courses that allows Bob to satisfy all the prerequisites. Use the topological sort algorithm.

12. Let e be a maximum-weight edge on some cycle of $G = (V, E)$. Prove that there is a minimum spanning tree of $G' = (V, E - \{e\})$ that is also a minimum spanning tree of G. That is, there is a minimum spanning tree of G that does not include e. Firstly consider the graph G'. Compute the MST of G', call it T'.

13. In this problem, we give pseudocode for different algorithms. Each one takes a graph as input and returns a set of edges T. For each algorithm, you must either prove that T is a minimum spanning tree or prove that T is not a minimum spanning tree. Also describe the most efficient implementation of each algorithm, whether or not it computers a minimum spanning tree.

Algorithm 13.12 Maybe Minimum Spanning Tree A

```
function MAYBE-MST-A(G, w)
    sort the edges into decreasing order by weight
    T ← E
    for all edge e, taken in decreasing order by weight do
        if T − {e} is a connected graph then
            T ← T − {e}
        end if
    end for
    return T
end function
```

Basic idea here is to start MST T containing all edges of the graph G, and removing edges in largest first order, making sure that we keep the graph G connected. This will create a valid MST. Implementation with edges in descending sorted order, and to check for connectivity as removing edges using a basic depth first of breadth first search to make sure graph is still connected.

Algorithm 13.13 Maybe Minimum Spanning Tree B

```
function MAYBE-MST-B(G, w)
    T ← ∅
    for all edge e, taken in arbitrary order do
        if T + {e} has no cycles then
            T ← T + {e}
        end if
    end for
    return T
end function
```

Algorithm 13.14 Maybe Minimum Spanning Tree C

```
function MAYBE-MST-C(G, w)
    T ← ∅
    for all edge e, taken in arbitrary order do
        T ← T + {e}
        if T has a cycle c then
            e′ ← the maximum weight edge on c
            T ← T − {e′}
        end if
    end for
    return T
end function
```

Basic idea here is to start with an empty MST T and in no particular order, add edges to it. If a cycle is formed in T, remove the maximum weight edge from the cycle in T. As long as you continue for all edges, this will create a valid