COSC242 Exam Prep and Misc Stuff Doc

Willem said going to over the last few years of exams should be enough.

*“there is probably going to be a question on* ***Prims algorithm****”* (min spanning tree)

The dynamic programming question is *“probably”* going to be **Assembly Line Scheduling**

Willem is available in his office every weekday excl.Monday; he usually has meetings in the morning.

Was this said in 2014 or 2013?

2013

[Countdown timer](http://csnet.otago.ac.nz/rcrimp/timer/index.html)

[flash cards](http://www.cram.com/cards/cosc242-3527221) - thanks amy :)

# Authors

John Smith, Anon Ymous, Peter Pan, Little John, Papa Smurf, and the seven dwarves→Ctrl+Alt+0 (and everyone else is welcome to use that format, too!)

Phil: My format: Coloured name, normal writing black. Best of both worlds! :D

Reuben: will sign

Amy (I have cram.com flashcards - kewlgrl5014)

*Logan*

Kadin

**LUCY**

Angus  **Chan**

**Mitchell**

**Alan**

**Ross**

**Gary**

Ella

Tavita

Gene  
Scotty

**Putin**

**Henry**

***ryan***

*Tori*

Chris

Batman

arteries

***The Flash***

aie-sha

# Algorithm breakdown

Grey: Most Likely **NOT** in the exam, but still good to know.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sorting Alg** | **Best** | **Average** | **Worst** | **in-place(buffer)** | **stable** |
| Insertion | O( n ) | O( n2 ) | O( n2 ) | yes | yes |
| Selection | O( n2 ) | O( n2 ) | O( n2 ) | yes | yes |
| Merge | O( n log n ) | O( n log n ) | O( n log n ) | no | no\* |
| Quick | O( n log n ) | O( n log n ) | O( n2 ) **rare** | yes | no\* |
| Counting | O( n ) | O( n ) | O( n ) | no | yes |
| Radix | O(n) | O(n) | O(n) | no | yes |
| Bubble | O( n ) | O( n2 ) | O( n2 ) | yes | yes |
| Bucket |  | O( n + k ) | O( n2 ) | no | yes |
| Heap | O( n log n ) | O( n log n ) | O( n log n ) | yes | no |

\* generally the case, but it depends on the implementation.

Stable: respects the original order of entries with equal keys.

In-Place: sorts the unsorted list within the array; ie it doesn’t copy itself and sort somewhere else.

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Structure** | **Insertion** | **Deletion** | **Search** |
| perfectly balanced BST | O( log n ) | O( log n ) | O( log n ) |
| **worst** unbalanced BST | O( n ) | O( n ) | O( n ) |
| Red-black Tee (RBT) | O( log n ) | O( log n ) | O( log n ) |
| Perfect Hash | O( 1 ) | O( 1 ) | O( 1 ) |
| Array | 1 | O( n ) | -- |
| Linked List | O(1) or O(n) | O( n ) | O( n ) |
| B-Tree | O( log n ) | O( log n ) | O( log n ) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Traversals** | **BFS** | **DFS** | **Prims** | **Dijkstra's** |
| **Graph type** | unweighted | unweighted | weighted edges | weighted edges |
| **Queue** | queue | stack | min priority Q | min priority Q |
| **Values found** | distance | discovery / finish | edge weights | cumulative edge weights |
| **Purpose** | Shortest path. | Connected components.  Topological sort of DAGs | Min spanning tree. | Single source cheapest path. |

|  |  |  |
| --- | --- | --- |
| **Hashing stuff** | **Description** | **Hash formula** |
| Collision →  Linear Probing | try to insert **k** at index **i**. If **i** is full try **i + 1**, repeat. | H(k,c)=(h(k) + c)%m h(k) = i |
| Collision →  Quadratic Probing | try to insert **k** at index **i**. If **i** is full, try **i + c2**, where **c** is the number of collisions, repeat. | H(k,c)=(h(k) + c2)%m  h(k) = i |
| Collision →  Double Hashing | try to insert **k** at index **i**. If **i** is full, try **i** + **c\*j** where **c** is the number of collisions, and **j** is the value from the second hash function. | H(k,c)=(h(k) + c\*g(k))%m  h(k) = i  g(k) = j |
| Collision →  Cuckoo Hashing | Insert at **i**. If **i** is full, push the value that was in **i** to it’s second hash. Basically, kick things from **h(k)** ←→ **g(k)**, until we get our thing done. | h(k) or g(k) |
| Collision →  Universal Hashing | fancy prime math ensures that more than half the time there isn’t a single collision with n elements in a table of size n2 | hab(k) = ((a.k + b)%p)%m |
| Chaining →  Linked List, BST... | elements that collide at index **i** are put In a data structure that *hangs* off index **i**. |  |

|  |  |  |
| --- | --- | --- |
| **Case** | **Features** | **Solution** |
| **Insertion Case 1** | Red Uncle | Push black into its children |
| **Insertion Case 2** | Black uncle, Red Inner Child | Rotate on parent and make into Case 3 |
| **Insertion Case 3** | Black Uncle, Red outer child | Swap colour of x’s parent and grandparent and rotate by picking up x’s parent |
| **Deletion Case 1** | X’s sibling is red | Lift w and swap colour with its parent (which will be it’s child after lifting) **(turns into 2, 3 or 4)** |
| **Deletion Case 2** | X’s sibling is black and it has two black children | Take one black from x and w and push it to their parent |
| **Deletion Case 3** | X’s sibling is black and its inner child is red and outer child **black** | Swap colour of w and inner child and pick up inner child (converts to case 4) |
| **Deletion Case 4** | X’s sibling is black and its’ outer child is red (inner child is black or red) | Swap colour of w and its red outer child and pick up w to put it on the path through x |

|  |  |
| --- | --- |
| **Examinable Subjects** | **Marks** |
| Identify time complexities of known algorithms? | (5-10) |
| Proofs: induction, iteration and contradiction | (8-15) |
| Sorting: the algorithms, time complexities, in-place, stable | (8-15) |
| Hashing: open addressing(linear, quadratic, double, cuckoo), chaining, perfect hash | (15-20) |
| Trees: BST, RBT and B-Tree | (15-25) |
| Graphs: representations, DFS, BFS, prims, dijkstras | (15-25) |
| Dynamic Programming: step through a problem | (5) |
| P, NP and NP-complete | (5) |

Section 1:

* Insertion and deletion of B-Trees, Prim’s.
* Dijkstra’s may be tested? I see no reason why not. - B

Section 2:

* Look back at first assignment.
* what does big-O mean?
* how to use induction to show big-O?
* iteration to find formula
* use contradiction to show that it is not big-O.

Section 3:

* How to step through each sorting: merge, linear, and quick sort;
* complexities;
* properties that sorting algorithms have;
* which sorting to use in which application;
* what is stability and example of why we care about stability;
* how to improve n log n;
* what extra information is needed to make linear sort more.

Section 4:

* All forms of open addressing;
* take keys and apply approaches to put them in given table;
* test to apply to hash function to see if it is good enough.

Section 5:

* Build trees,
* use deletions/insertions;
* questions about peripheral stuff;
* traversals; used to find maxima and minima.

Section 6:

* What two representations graphsand their pros/cons;
* applications of DFS such as double logical sorting;
* minimum spanning trees;
* difference between Dijkstra’s and Prim’s;
* complexity of DFS and BFS (BFS: all vertices white, grey when in queue, then black, distance value; DFS: discovery time/ending time)

Section 7:

* Show you understand how to use dynamic programming from lecture examples (assembly),
* Fill out two tables and show how to get work in assembly-line.

Section 8:

* What problems can be verified by which;
* what are the classes;
* why do we care?
* examples of problems in each class;
* NP-complete problem to identify.

# Useful Links

<http://www.cs.otago.ac.nz/cosc242/> COSC242 Home Page

<http://www.otago.ac.nz/library/exams/> Get past exam papers here

<http://faculty.cs.niu.edu/~freedman/340/340notes/340hash.htm> Hashing tute/explanation

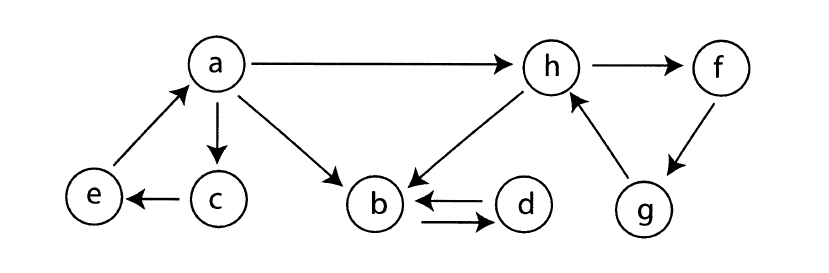
<http://www.cse.yorku.ca/~aaw/Sotirios/RedBlackTree.html> RBT’s

<http://www.drdobbs.com/cpp/why-stable-sorting-is-important/231001864> stable sorting purpose

<http://www.sorting-algorithms.com> Sorting Algorithms

[DFS](http://courses.csail.mit.edu/6.006/fall10/handouts/quiz2review.pdf)   
<http://bigocheatsheet.com/> Big-O Cheat Sheet

## Random Questions



**Perform a breadth-first search on this graph starting from ‘a’. Include the queue and final distances.**

B: *Taken/adapted from Lecture 18 Slide 6:*

For every vertex in G that isn’t source:

set colour[u] ← white, dist[u] ← ∞

colour[source] ← grey and dist[source] ← 0

Add source to our queue, Q

while Q not empty:

dequeue curr from Q

for each node adjacent to curr:

if colour[adjacent] == white:

colour[adjacent] ← grey

dist[adjacent] ← dist[curr] + 1

Add adjacent to Q

colour[curr] ← black

*So based on that algorithm:*

*YOU MIGHT WANT TO KEEP TRACK OF THE PREDECESSORS*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node | a | b | c | d | e | f | g | h |
| Colour | grey | white | white | white | white | white | white | white |
| Distance | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |

We add the source node to our queue; Q = {a}.

Take a from Q: we look at its neighbours; [c, b, h] and, if their colours are white, update their colour and distance accordingly. As we do this, we also add them to the queue:

Q = {b, c, h}.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node | a | b | c | d | e | f | g | h |
| Colour | black | grey | grey | white | white | white | white | grey |
| Distance | 0 | 1 | 1 | ∞ | ∞ | ∞ | ∞ | 1 |

Take b from Q: look at its neighbours: [d]. Update:

Q = {c, h, d}

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node | a | b | c | d | e | f | g | h |
| Colour | black | black | grey | grey | white | white | white | grey |
| Distance | 0 | 1 | 1 | 2 | ∞ | ∞ | ∞ | 1 |

Take c from Q: look at its neighbours: [e]. Update:

Q = {h, d, e}

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node | a | b | c | d | e | f | g | h |
| Colour | black | black | black | grey | grey | white | white | grey |
| Distance | 0 | 1 | 1 | 2 | 2 | ∞ | ∞ | 1 |

Take h from Q: look at its neighbours: [b, f]. Update:

Q = {d, e, f} // Note b is black so we don’t add it to the queue etc

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node | a | b | c | d | e | f | g | h |
| Colour | black | black | black | grey | grey | grey | white | black |
| Distance | 0 | 1 | 1 | 2 | 2 | 2 | ∞ | 1 |

Take d from Q: look at its neighbours: [b]. Update:

Q = {e, f}

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node | a | b | c | d | e | f | g | h |
| Colour | black | black | black | black | grey | grey | white | black |
| Distance | 0 | 1 | 1 | 2 | 2 | 2 | ∞ | 1 |

Take e from Q: [a]. Q = {f}.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node | a | b | c | d | e | f | g | h |
| Colour | black | black | black | black | black | grey | white | black |
| Distance | 0 | 1 | 1 | 2 | 2 | 2 | ∞ | 1 |

Take f from Q: [g]. Q = {g}.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node | a | b | c | d | e | f | g | h |
| Colour | black | black | black | black | black | black | grey | black |
| Distance | 0 | 1 | 1 | 2 | 2 | 2 | 3 | 1 |

Take g from Q: [h]. Q = {}. Done.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Node | a | b | c | d | e | f | g | h |
| Colour | black | black | black | black | black | black | black | black |
| Distance | 0 | 1 | 1 | 2 | 2 | 2 | 3 | 1 |

Now we can draw the tree JUST KIDDING

**Find the minimum spanning tree of the graph above, by using Prim’s algorithm. Starting from ‘a’. Include the min priority queue and the minimum tree that results from this.**

the Tree be like:

( a )―2―( c )―3―( f )―1―( g )―2―( d )―2―( e )―1―( b )

<http://en.wikipedia.org/wiki/File:Red-black_tree_example_%28B-tree_analogy%29.svg>   
help!

## NP, P, NP-Complete



P: Sorting, Hashing

NP: factoring primes

NP-Complete: Hamiltonian Cycle Problem

solvable but not any of the above: NxN Chess

# Exam Questions from Past Papers

**2013**

**1. Complexity classes**

**(a) How many comparisons will Mergesort perform on an input array of length *n*? (1)**

**O(nlog(n))**

**(b) What is the worst case time complexity of Quicksort on an input array of length *n* assuming we use our unmodified Partition algorithm? (1)**

**O(n2)**

**(c) What is the worst case time complexity of Binary Search on a sorted input array of length *n*? (1)O(log(n))**

**(d) What is the worst case time complexity of searching a linked list of length *n*? (1)**

**O(n)**

**(e) What is the worst case time complexity of searching a Red-Black Tree with *n* keys? (1)**

**O(log(n))**

**(f) How many distinct subsets does a set of size *n* have? (1)**

**2n**

**(g) How many permutations may be generated from a set of *n* items? (1)**

**n!**

**2. Recurrences, Big-O, Induction**

**(a) Show by using the iteration method that the recurrence equations**

**T(1) = 1**

**T(n) = 2T(n/2) + n**

**define the function given by T(n) = n log n + n.**

**You do not need to prove that your solution is correct. You may assume values of n are powers of 2. (5)**

**T(n) = 2T(n/2) + n**

**= 4T(n/4) + n + n**

**= 8T(n/8) + n + n + n**

**?= 2kT(n/2k) + kn (assuming n = 2k)**

**= nT(1) + log2(n) . n** can someone explain why they’ve taken k = log(n) here?

I think its cuz k = log2n which is reverse of n = 2^k

**= n log(n) + n**

**(b) Using the definition of big-O and induction, prove that n2 = O(2n). (6)**

**f(n) = O(g(n)) if and only if there exists some constant A and some value l such that f(n) <= Ag(n) for all n >= l**

**Let A = 1, l = 4**

**42 <= 1.24**

**16<= 16 true.**

**if n2 <= 2n then (n+1)2 <= 2(n-1)  I think you mean (n+1)2 <= 2(n+1)**

**(n+1)2 = n2 + 2n + 1**

**<= 2n + 2n + 1 (Using induction hypothesis)**

**<= 2n + 2n + n (Because n>=4)**

**<= 2n + 3n**

**<= 2n + n2 (Because n2 = n.n and 3<n because n>=4)**

**<= 2n + 2n (Using induction hypothesis)**

**<= 2.2n**

**<= 2(n+1)**

**Therefore if n satisfies the condition so does n+1.**

**(c) Use proof by contradiction to show that n3 O(n2). (2)**

**If n3 is O(n2) then there exists an A and l s.t. n3 <= A.n2 where n>=l**

**since n3 = n.n.n and n2 = n.n**

**n.n.n <= A.n.n**

**n <= A**

**obviously since n will always grow and A is a constant n will always grow beyond A therefore n3 /= O(n2)**

## 

## Sorting

## (a) Suppose you had to implement a sorting algorithm under conditions where memory is tight. Which of the following sorting algorithms could you use? Insertion Sort? Mergesort? Quicksort? (2)

Insertion sort would be the best choice as it only uses O(1) auxiliary space (it is in place). Quicksort can also be in place but because of the recursive calls it uses O(logn) auxiliary space. Mergesort would be the worst choice as it uses O(n) extra memory because it is not an in place sort.

What’s your source on Quicksort and aux space? Do we need to know it? because I don’t remember anything about that being in the lectures.

In the lectures all it says is that both insertion sort and quicksort are in place (last slide of lecture 8) but the O(logn) comes from wikipedia http://en.wikipedia.org/wiki/Quicksort. The reason for the extra space is because the pointer A and two integers low and high (using pseudocode from slide 5 lecture 7) are held in the stack for every recursive call to Quicksort(A, low, high) which means there will be logn different copies of A and logn low and high variables at the deepest level of recursion. In the worst case there will be n copies if the data is sorted and we drop into the O(n2) case as one call to quicksort will go n levels deep and the other will only go one deep. Insertion sort on the other hand will only have one copy of A, one copy of n and two integers i and j.

P

In answer to whether we need to know it, we probably only need to know that mergesort isn’t in place and insertion sort and quicksort are. < Cool cheers

## (b) Now suppose you discover that your sorting algorithm would often be given nearly-sorted input. Which algorithm would you implement and why?(2)

## On nearly sorted inputs Insertion sort will run in almost O(n), mergesort will still run in O(nlogn), and Quicksort will run in O(n2) depending on the pivot that is chosen. Even if a median of three is used and gives a good partition Quicksorts best case is O(nlogn) so Insertion sort is the best choice for this instance.

## (c) Name one algorithm that can sort integers in O(n) time. What requirements must be satisfied in order for this algorithm to be used and to work efficiently? (3)

Counting sort can sort integers in O(n) time. In order for it to be efficient the difference between the maximum and minimum keys cannot be much larger than the number of keys to be sorted.

Both Bucket sort and radix sort can sort in O(n) provided the algorithm used to sort the buckets runs in O(n), So if the buckets are small and/or likely to be nearly sorted, insertion sort can be used or Counting sort can be used.

## (d) Use Bucket Sort to sort the following keys: 0.92, 0.55, 0.86, 0.13, 0.52, 0.88, 0.25. (3)

## Buckets:

0.0x 0.1x 0.2x 0.3x 0.4x 0.5x 0.6x 0.7x 0.8x 0.9x

0.13 0.25 0.55 0.86 0.92

0.52 0.88

Sort Buckets:

0.1x 0.2x 0.5x 0.8x 0.9x

0.13 0.25 0.52 0.86 0.92

0.55 0.88

Join Buckets:

0.13, 0.25, 0.52, 0.55, 0.86, 0.88, 0.92.

For bucket sort don’t you use the number of items as the number of buckets? ~~So you would actually have only 7 buckets and multiply each key by 7 to find out which bucket the key belonged in.~~

~~(~~**~~0.92~~**~~\*7 =~~ **~~6~~**~~.44;~~ **~~0.55~~**~~\*7 =~~ **~~3~~**~~.85;~~ **~~0.86~~**~~\*7 =~~ **~~6~~**~~.02;~~ **~~0.13~~**~~\*7 =~~ **~~0~~**~~.91;~~ **~~0.52~~**~~\*7 =~~ **~~3~~**~~.64;~~ **~~0.88~~**~~\*7 =~~ **~~6~~**~~.16;~~ **~~0.25~~**~~\*7 =~~ **~~1~~**~~.75)~~

~~0 1 2 3 4 5 6~~

~~0.13 0.25 0.55 0.92~~

~~0.52 0.86~~

~~0.88~~

**In the lecture slides he uses the buckets by multiplying the numbers by 10 to make them integers e.g. 0.17 x 10 = 1.7 so goes under bucket number 1. But you could be right. that’s just the example he uses.**

^The example he uses n=10, but the notes (L9 slide 6) mentioned taking the input array of size n and then creating an auxiliary array of length n, off which hang the linked lists. So that’s where I got that from.

that makes much more sense :)

^Spoke to Willem: Just use 10.

sweet

## (e) What is the difference between a stable sort and an unstable sort? Which of the following sorts is stable? Insertion Sort? Quicksort? Counting Sort? (3)

## A stable sort means that items with the same key value will have the same position relative to each other at the beginning and the end of the sorting. Insertion sort and counting sort are both stable but quicksort isn’t

## 

## Hash Tables

1. Given a table of size 7, a hash function h(k), and input keys 86, 96, 93, 34, 11, 26, 73, draw the hash table that results from:

(i) Open addressing with double hashing. Use h(k) = k%7 as the primary hash function, and g(k) = 1 + (k%6) as the secondary hash function.

Use primary hash function to find what index the key should go into. If that index is occupied, then use the secondary hash function instead, and place your key in the index x places to right of the occupied spot, where x is the value obtained from the secondary hash function.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 73 | 26 | 86 | 11 | 34 | 96 | 93 |

he gives us an equation in the lecture notes for double hashing

are we expected to use the equation to do double hashing?

H(k) = (h(k) + collisions \* g(k))%m you mean this?

yes!

## Chaining, with universal hash function h(10,5)(k) = ((10k +5)%101)%7.

|  |  |
| --- | --- |
| 0 | 96 34 11 26 73 |
| 1 | 86 |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 | 93 |
| 6 |  |

^^I had 86 hashed at position 1...Oops fixed.

1. Cuckoo Hashing with h(k) = k%7 as the primary hash function, and h(10,5)(k) = ((10k +5)%101)%7 as the secondary.

|  |  |
| --- | --- |
| 0 | 96 |
| 1 | 86 |
| 2 | ~~86~~ 93 |
| 3 | 73 |
| 4 | 11 |
| 5 | ~~96~~ 26 |
| 6 | 34 |

1. Suppose you were using a perfect hashing scheme to create a hash table from the keys above. Would h(10,5) be acceptable as the primary hash function? Give your reasoning.(2)

can someone answer this?

No the function h(10,5)(k) given the keys above hashes over half of them to position zero. This is not ideal as this will result in many collisions breaking down the O(1) property of hash table insertion and searching.

could you use the equation: ni^2 <2n where n is the number of positions and ni is the number of keys at each position. So a good hash function should be ni^2 <2n.

so you work out what ni^2 is and compare it to 2n. If it’s smaller its a good hash function?

Yes you can use that equation.

is that what we are suppose to use?

Spoke to Willem about this question, and he used that equation. Unsure but I guess so? Sorry if not being helpful~ :/

So;ni22n (i.e. the sum of all occupied indices squared).

ni2 2.7 = 27 14, which is not true, so it is not acceptable as the primary hash function.

n0 = 5

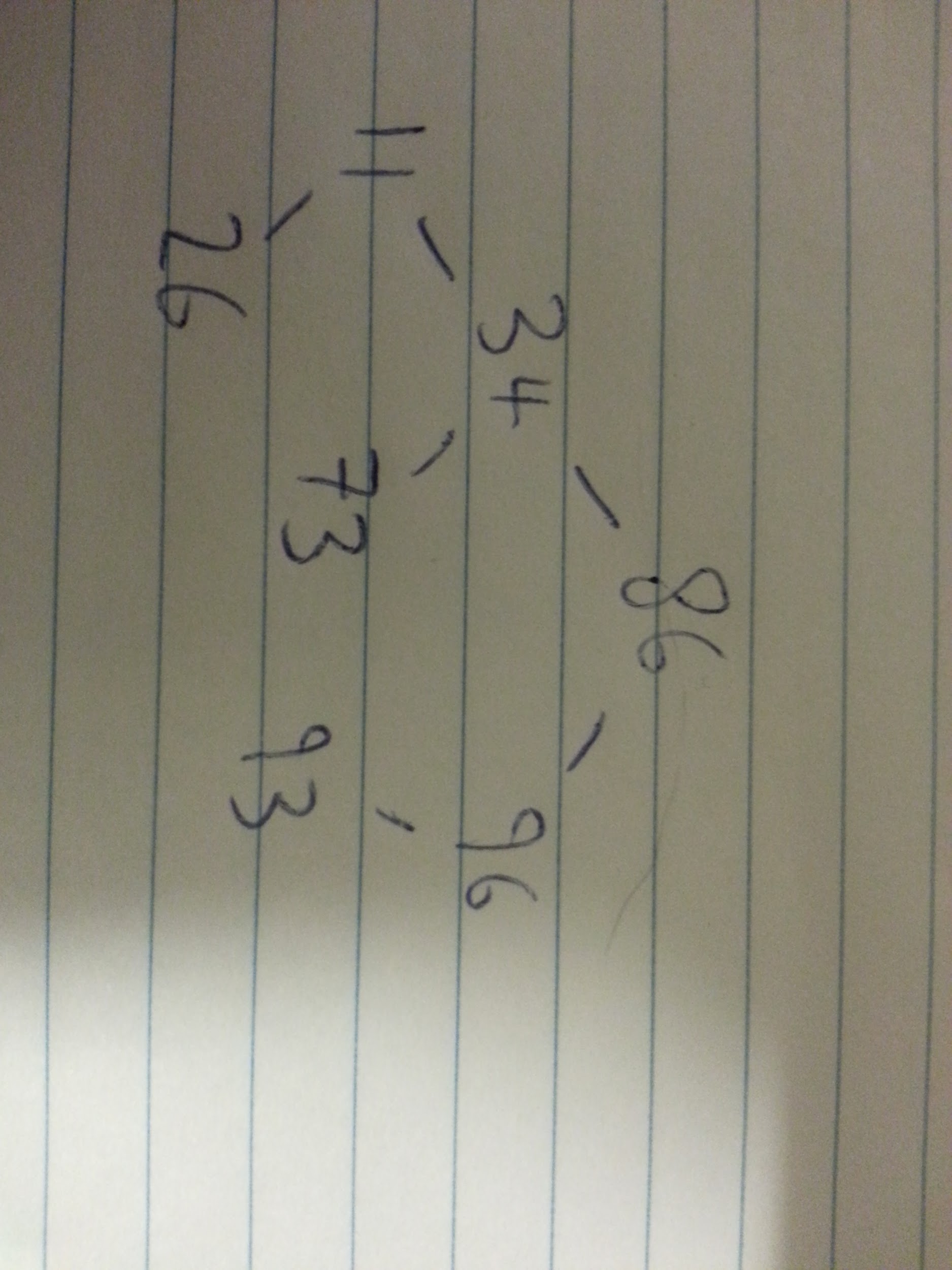
n1 = 1

n2, n3, n4 = 0

n5 = 1

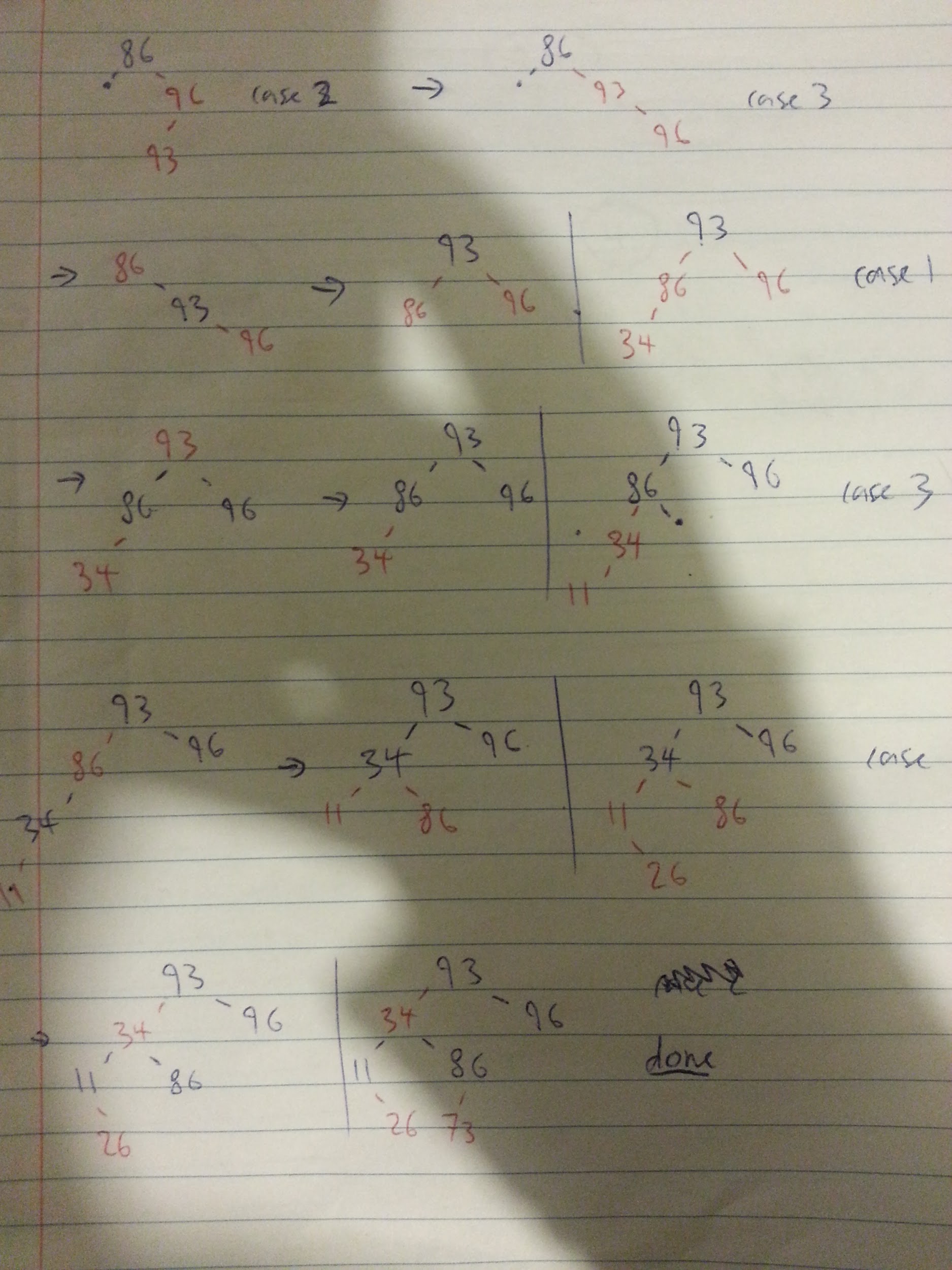
n6 = 0

So 52 + 12 + 12 = 27

1. **Trees**
2. Draw the final BST that results from inserting 86, 96, 93, 34, 11, 26, 73.
3. Write down the keys of the tree in the order they would be visited during a postorder traversal.

**26, 11, 73, 34, 93, 96, 86.**

1. Show the RBT that results by inserting 86, 96, 93, 34, 11, 26, 73. State which cases apply.



1. Some help with this one? Deletion of 96

You do a case 1 then a case 4 I think. i.e. you end up with (Sorry if i stole someones colour)

34 (b)

/

11 (b) 86 ®

\ / \

26(r) 73 (b) 93 (b)

could you / someone please explain this in a few more steps, still not sure, and the lecture notes dont explain it that well.



Not quite, you DON’T need to split up the [ 2, 4, 6 ] node.

In the lecture notes they had the same situation, except they were about to add 10 afterwards, in which case you would start splitting nodes up.

(f)



can someone confirm this?

Pretty sure via the one pass strategy you need to merge 2, 4, 7 as its too thin. i.e. you get:

2, 4, 7

/ | | \

1 3 6 8, 9

^Yup, I got the same.

6. Graphs

(a) Give one reason why you might prefer to use an adjacency matrix representation of a graph, and one reason why you might decide to use an adjacency list

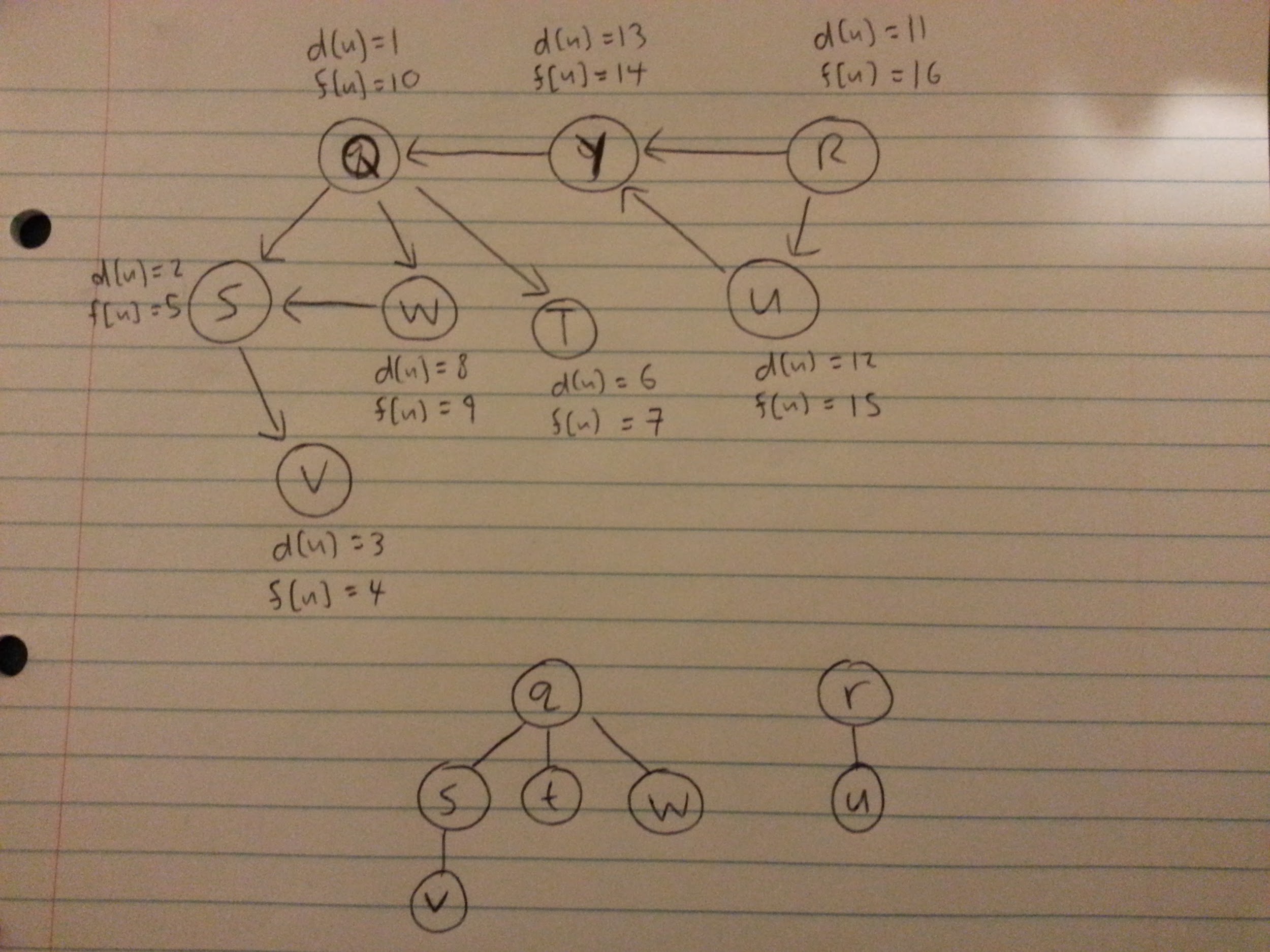
representation. (4)

Depending on the operation you want to use on the graph on representation of the graph edges is stronger than the other. Provided you have a dense graph with the number of edges closer to n2 (where n is the no. vertices) than n, and you want to determine whether two vertices are linked, then the space cost of the adjacency matrix may be worth the O(1) operation time. However for sparser graphs a linked list representation is more space efficient and performs better for finding adjacent vertices and traversals.

^Don’t think it needs to be this verbose.

(b) Copy the following directed acyclic graph into your answer book. Starting at q, and considering adjacency lists to be alphabetically ordered, show how depth-first

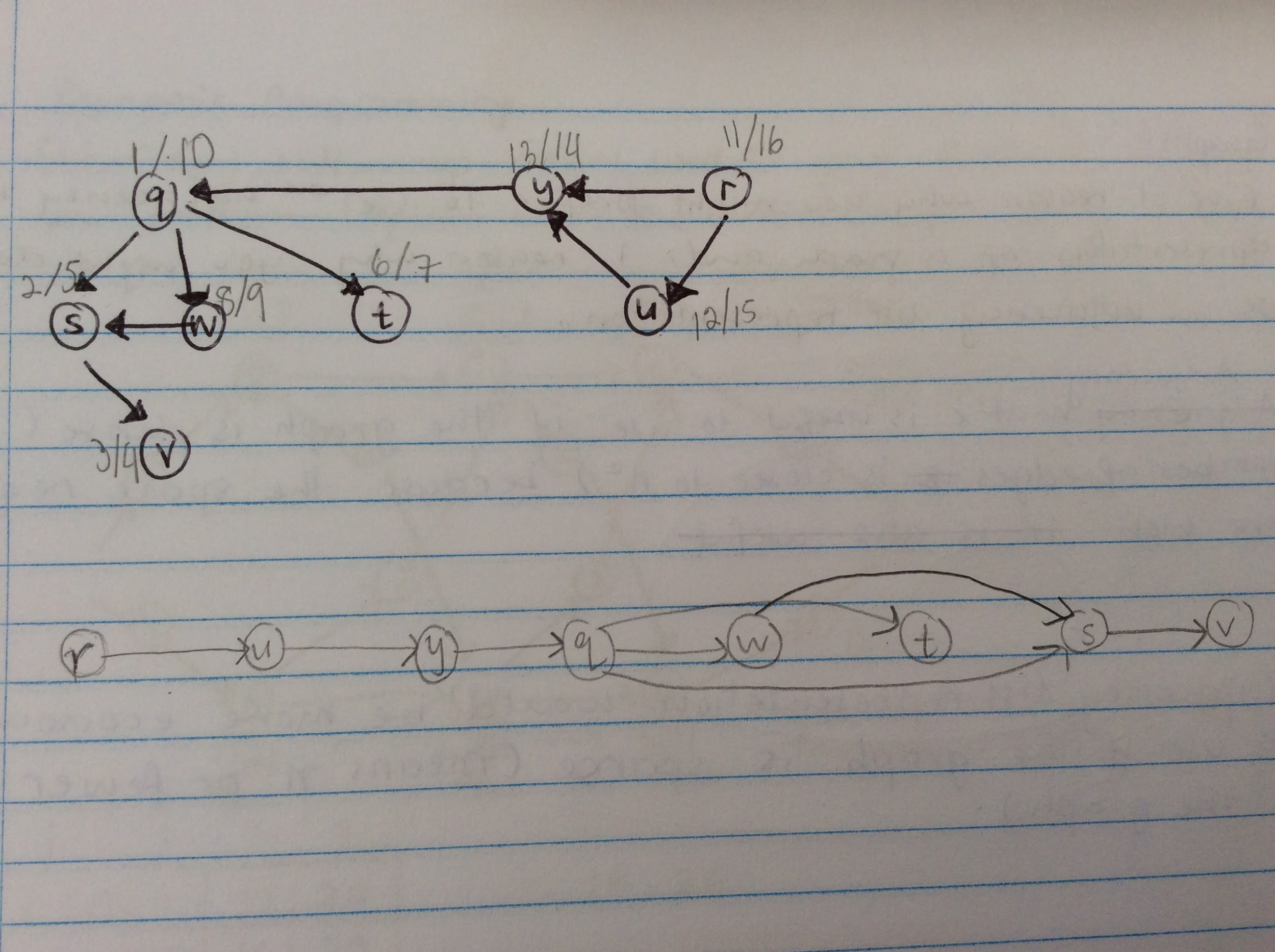
search would allocate time stamps to vertices, and then topologically sort the data. (8)



Is this right in the slightest?

yeah thats what I get but I’m kinda confused by the topological sort part. I thought that was sorting it from left to right with biggest finishing times to smallest. Are we supposed to be them into trees? i thought it was a list? i’m prob wrong though

Hey, so the DFS looks correct there, that’s what I got too (photo below). As for trees, no, we do not have to draw them (the question did not say so and we won’t get extra marks for doing so). The topological sort data is like what Tori mentioned: do it from left to right with the biggest finishing times to the smallest. And yep, I checked with Willem too. ~~btw, w is also connected to s (sorry for the mistake).~~  updated photo below:



should w be connected to s as well?

whops, yes, it should be. w->s. (updated photo above).

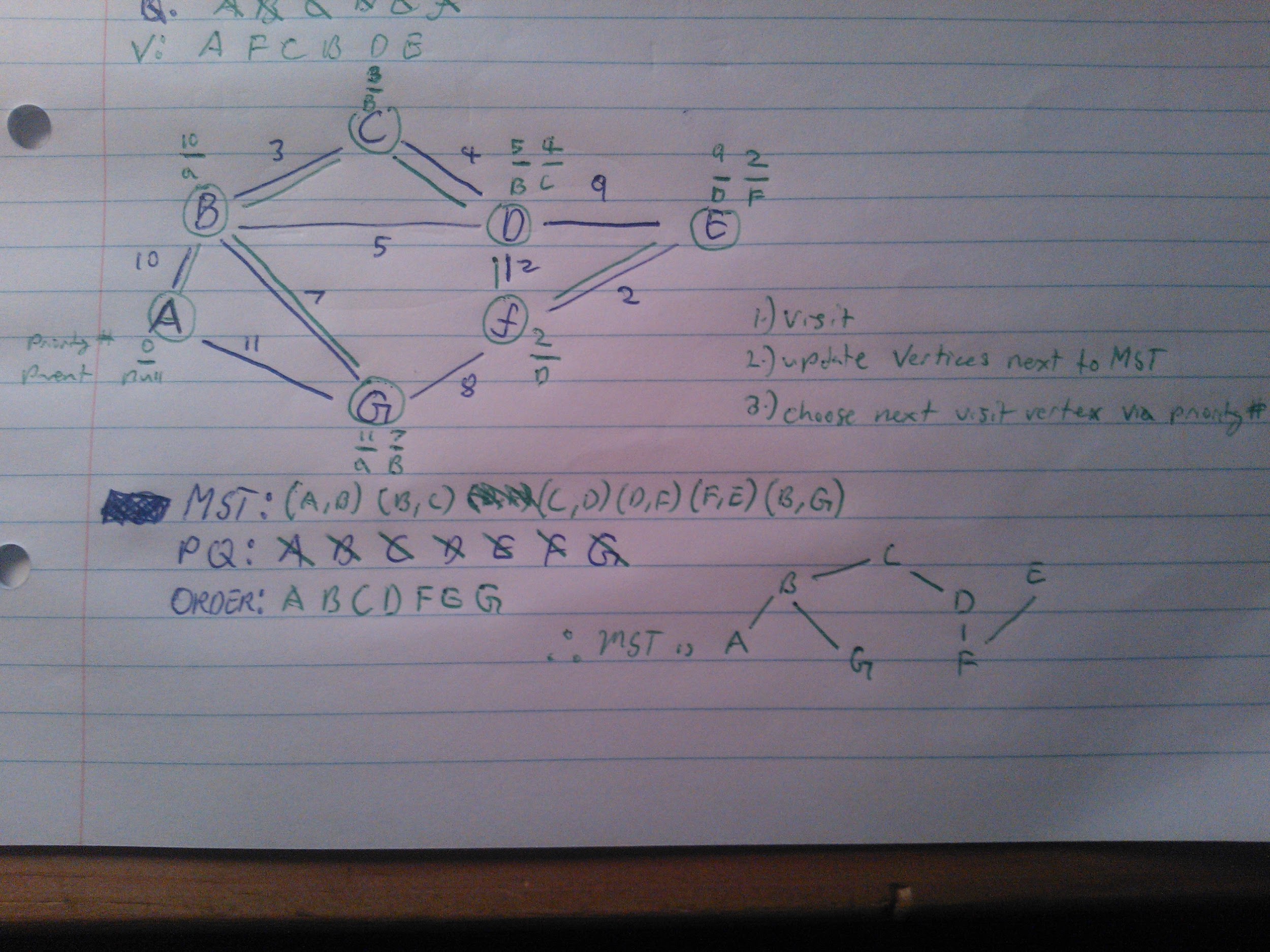
(c) Copy the following weighted undirected graph into your answer book. Show how Prim’s algorithm would find the minimal spanning tree with root a. Show clearly how the priority values change, and show the order in which vertices are extracted from the priority queue. Give a table showing vertices and their parents from which the tree can be computed. (8)

can someone answer this?

I answered this using a youtube tutorial…

<https://www.youtube.com/watch?v=z1L3rMzG1_A>

this is what i got



7. Dynamic programming

Consider the assembly line scheduling problem below. Give a dynamic programming solution. Show any bottom-up tables used in your solution and any calculations you perform. Explain what the entries in your tables mean. (5)

can someone help explain this question :)

8. P and NP

In a few well-chosen sentences, explain to Aunt Maud what the classes P and NP are, and what it means to say that a problem is NP-complete. Give her one example of an NP-complete problem. (5)

P is the class of all solvable problems solvable by algorithms in polynomial time. NP, however is the class of all solvable problems for which a solution may be checked in polynomial time. This does not necessarily mean that the problem is easy to solve. A problem is referred to as NP-complete when it is the class NP and every other problem in the class NP can be easily rephrased as a variation of that problem, meaning that an algorithm which solves one NP-complete problem can be modified to solve all NP-complete problems. An example of an NP-complete problem is the Hamiltonian Cycle Problem (HCP).

^Probably a better answer to this in prev. years.

2012

### 1: Complexity Classes

#### a) How many comparisons will Insertion Sort do if the input is an already-sorted array of length n?

**n-1 comparisons.**

#### b) What is the worst case time complexity of Insertion Sort on an input array of length n?

**O(n2)**

#### c) What is the worse case time complexity of Mergesort on an input array of length n?

**O(n log n)**

#### d) If you were given an input array of small integers ranging in value from 0 to 113, and the length of the array was 100, what would be the most efficient algorithm with which to sort the input?

**Counting sort.**

Because the range of values is known (ie keyspace is known) and the keyspace is about the same size as the array length (keyspace = O(array length) )

what is counting sort?

This <http://www.youtube.com/watch?v=3mxp4JLGasE&t=15m53s>

#### e) What is the worst case time complexity of searching an unsorted array of length n?

**O(n)** - You have to go through the entire array.

#### f) What is the worst case time complexity of Binary Search on a sorted input array of length n?

**O(log n)**

B: “Binary search” → we cut the available search space in half with each comparison. So ya, O(log2(n)).

^this doesnt imply BST, just binary search

#### g) What is the worst case time complexity of searching a Red-Black tree with n real nodes.

**O(log n)**

B: Is this because, due to the properties of an RBT, we know that the tree is balanced within one level; searching is O(log2(n))?

#### h) How many distinct subsets does a set of size n have?

**2n**

What is this asking?

Phil: “How many times can you break up a set?”

B: [different ways] ^uniquely sorry for hue

* From the empty set ∅ := {}, you can only choose one set; itself.
* From the set consisting of one element, say {x}, you can choose {x} or the empty set, ∅.
* From the set consisting of two elements, say {x, y}, you can choose {x, y}, {x}, {y}, or ∅.

By observation we can guess that a set of n elements has 2n distinct subsets. You could prove this by induction. (Sorry for interrupting your comment Phil)

Phil: The set {1,2} can be broken as follows…

{} - the empty set (it counts right?) B: yes

{1}

{2}

{1,2} - I’m pretty sure something can be a subset of itself?

Which is 4 sets, or 2^2 sets

And another example. The set: {1,2,3} can be broken as follows…

{} (the empty set)

{1}

{2}

{3}

{1,2}

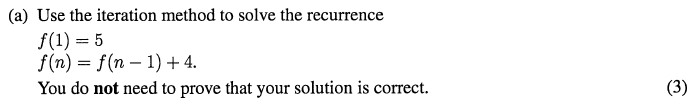
{1,3}

{2,3}

{1,2,3}

Which is 8 sets, or 2^3.

### 2: Recurrences, Big-O, Induction



f(1) = 5

f(n) = f(n-1) + 4

f(n-1) = f(n-2) + 4

so f(n) = (f(n-2) + 4) + 4

= f(n-2) + 8

f(n-2) = f(n-3) + 4

so f(n) = f(n - 3) + 4 + 4 + 4

= f(n - 3) + 12

We can see a pattern emerging.

For every number k subtracted from n in (n - k) , the number added on is 4 \* k.

Therefore the pattern is:

f(n-k) + 4k

We need to remove f(n-k), so we choose k, such that f(n-k) = f(1) i.e k = n - 1

So now we have f(n) = f(n - (n - 1)) + 4 ( n -1 )

= f(1) + 4n - 4 Note: f(1) = 5

= 4n + 5-4 = 1+4n

= 4n + 1

f(n) = 4n + 1



if n2 = O(n!) then there exists c and n0 such that n2 ≤ c \* n! for all n > n0.

Let’s c = 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| n | 1 | 2 | 3 | 4 | 5 |
| n2 | 1 | 4 | 9 | 16 | 25 |
| n! | 1 | 2 | 6 | 24 | 120 |

16 ≤ 24

So n2 ≤ n! when n = 4

**Induction hypothesis**: k2 ≤ k! (for k = 4)

now prove for k + 1: (k+1)2 ≤ (k+1)!

(k+1)2 = k2 + 2k + 1 (expanding, foil)

≤ k2 + 2k + k (since k ≥ 4, we can replace 1 with k)

≤ k2 + 3k

≤ k2 + k\*k (since k ≥ 4, we can replace 3 with k)

≤ k2 + k2

≤ 2k2

≤ 2\*k! (by our **induction hypothesis**, k2 ≤ k!)

≤ (k+1) \* k! (since k ≥ 4, we can replace 2 with (k+1)

≤ (k+1)!

Which is what we wanted, (k+1)! is exactly the same as the RIght Hand SIde

So we proved true for k (k = 4), we’ve proved true for k + 1, therefore it’s true for all k

Therefore n2 = O(n!)

B: Gave up on this, but this was where I got to:

-

So n2 is O(n!)

### 3: Sorting



Drop to Insertion sort on when subarrays reach a smaller size, because Insertion sort is faster than mergesort on small arrays.

Mergesort does not sort in place, instead the merge routine copies items into a new array. One way to improve Mergesort would be to reduce the amount of copying done to items in the new array by using another sort method such as insertion sort.

It is calculated in lectures to be more efficient to use Insertion Sort when the size is between 40 and 48, but it is common practice to drop to Insertion sort when size is about 20 to account for differences across machines, hardware etc.



Quicksort uses a pivot to sort. It takes (hopefully) the median value and the puts all of the values small to the median to the left of the median, and all of the values larger than the median to the right of the pivot. The algorithm is then called recursively to each side of the pivot, picking new pivots and putting the smaller values on the left and the bigger values on the right. If the pivot is bad, eg is consistently the smallest or largest value, then the time complexity O(n2). To get the true median requires a large overhead. A good compromise is to take the first, last, and middle values, and taking the middle one as the pivot. This improves partitioning, and thus ensures O(nlog(n))

How is that?

*I have the same ideas as you, I just worded it differently..*

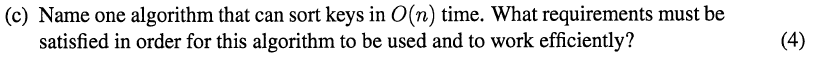
*In quicksort the array gets partitioned into 2 subarrays around the pivot value. Ideally (for best efficiency) the pivot should lie in the middle of the array. Some quicksort implementations take the value of the first element in the array for the pivot. While this is fast and simple, there is no guarantee that this value will be near the middle of the array. Taking the median of the entire array would give an accurate mid value, but is too time consuming. A good compromise is to take the median of the first, middle, and last elements. This is quick to perform and gives a reasonable representation of a middle value.*

B: It is important to meet the “well-planned” spec of the question: Introduce a problem, discuss its impacts, and then provide a solution (generally in that order so the paragraph makes sense). I.e., a poorly-chosen pivots wrecks Quicksort… we might want a better pivot selection algorithm. I hate sounding like a dick but I want to encourage people to try and refine their answers :).

S or T -- Topic - Also could be Statement ;)

E -- Explanation

E -- Example (or X... but I didn't put X)



Counting sort; but only when you know the range of keys and that range is similar to the number of keys in the array.

Radix sort would also be a good answer

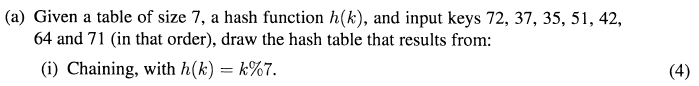
<http://www.youtube.com/watch?v=ibtN8rY7V5k>

But Bubble and Insertion also does it in O(n) if the keys are in order?

B: **Insertion/Bubble Sort is wrong**; it isn’t doing any sorting on the keys; these answers are edge cases and “gimmicky”. In fact, the absolute minimum sorting time for comparison-based sorts is O(n \* log2(n)) (on unsorted arrays). You should be mentioning **radix sort** and **counting sort**, and describing their preconditions:

Counting sort is when we maintain a count array of how many times a number has occurred and use this print out the list in order. This only works when we know the range of the values we are using e.g 0-100. We create an array thats the size of the range of values. So for 0-100 we would create a count array of size 101. Each index position is representative of a number in the range, and in the count array we store how many times that number/index number has occured. So if 2 arises 24 times in the array, at position 2 in the count array we store the number 24. When iterating through we would say print i has occurred a[i] times. Because no comparisons are made the algorithm simply traverses through the list once resulting in O(n) time.

### 4. Hash Table



Isn't chaining when you append them into the same hash position

I concur B: Chaining -- multiple values hashed into the same positions are appended at that position e.g. in a linked list or binary search tree. I agree with the table below.

|  |  |
| --- | --- |
| 0 | [35,42] |
| 1 | [64,71] |
| 2 | [72,37,51] |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |



|  |  |
| --- | --- |
| 0 |  |
| 1 |  |
| 2 | 42 |
| 3 | 71 |
| 4 | 37 |
| 5 | [35,51] |
| 6 | [72,64] |

looks like mine :)



Attempt to insert k at position h(k), if it’s occupied, the attempt to insert at h(k) + g(k), if that’s occupied then: h(k) + 2\*g(k)

so in general terms it’s h(k) + c\*g(k), where c is number of collisions, and you wrap around the hash table like it’s a circular array.

or H(k,i) = ((h(k) + i\*g(k))%m

|  |  |
| --- | --- |
| 0 | ~~42~~ ~~51~~ ~~71~~ 35 |
| 1 | ~~71~~ 64 |
| 2 | ~~37~~ ~~51~~ ~~42~~ 72 |
| 3 | 71 |
| 4 | ~~71~~ 42 |
| 5 | 51 |
| 6 | 37 |

Putting it in simple example its like when,

step = C - (k % C)

When there is collision it will go..

h(k), h(k) + step, h(k)+2\*step, h(k)+3\*step ....

For example when size M = 7, C is 5, and the key = 13

h(13) = 13% 7 = 6

h`(13) = 5 - (13% 5 ) = 2

First hashing function : h(13) = 6

After first collision we apply secondary hashing : (h(13)+h`(13))%7 = (6+2)% 7 = 1

After doing the step before and still have collision : (h(13) + 2 \* h`(13)) % 7 = (6+2\*2) % 7 = 3

If we keep trying this equation the answer will be increasing by 2 such as, 6,1,3,5,0,2,4..

It will go through every index of the table.

But if the table size is not a prime number such as 6 it will be like

1,3,5,1,3,5,1,3,5,.. Therefore the table size must be prime number.

So my answer is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | k%7 | (secondary) (10k%101)%7 | (first Collision)  (primary+secondary)%sizeOfTable | (Second Collision)  (primary+(secondary\*2))%sizeOfTable |
| 72 | 2 | 6 |  |  |
| 37 | 2(collision happens) | 4 | 6 |  |
| 35 | 0 | 5 |  |  |
| 51 | 2 | 5 | 0(collision happens) | 5 |
| 42 | 0 | 2 | 2(collision happens) | 4 |
| 64 | 1 | 6 |  |  |
| 71 | 1(collision happens) | 3 |  |  |

So my final hashtable will look like :

|  |  |
| --- | --- |
| 0 | 35 |
| 1 | 64 |
| 2 | 72 |
| 3 | 71 |
| 4 | 42 |
| 5 | 51 |
| 6 | 37 |

I got the same result as you.

**using 72 as example below**

/\*some variable declarations\*/

initial value = 72;

current value = initial value;

2ndHash(72)= 6; // because ((72\*10)%101) %7 = 6

for each value {

1. run the value through the primary hash function and try to insert value into the resulting index.

(72 % 7 = 2, so try to insert 72 at position 2)

2. If there is no value in that index already, insert this value.

3. If there IS a value: current value += 2ndHash(72); ie 72 += 6, so now our current value = 78;

please note: you increment the current value by 6 every time a collision occurs

4. Repeat all steps with value current value= 78, **IMPORTANT**: but remembering that when you find an empty slot, insert the initial value (72) into the hash table and not the current value 78.

^ geuh, i give up. theres probs something wrong with that.



Kadin: Firstly attempt to insert a value into the position designated by the primary hash function. If this position is free, no worries, we can insert it and leave it for now. If not: We examine the value that is in there. Is it in the position designated by the primary hash function (K%7)? or is it in the position designated by the secondary hash function(10k%101)%7 ?

Once we know this we need to move the value that was in there to make way for the new value coming in. So if a value is in its position that was designated by the secondary hash function, we simply move it to the position decided by the primary hash function and vice versa if it’s in its primary position. We follow this process each time we insert a value into the table.

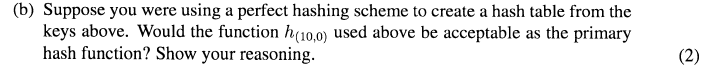
For example, say we insert 72 into the table, its position is 2, as decided by the primary hash function. Along comes 37, which also wants to be inserted at position 2. We look at 72, its primary hash function’s position is 2, so we know we need to move it to the position indicated by its secondary hash function. Now we have 37 at position 2, and 72 at position 6.

Sometimes we will swap numbers in and out of the same position multiple times, but eventually it works out, and each number will have its own position.

|  |  |  |
| --- | --- | --- |
| k | h(k) = k=k%7 | (10k%101)%7 |
| 72 | 2 | (720 % 101)%7=6 |
| 37 | 2 | (370%101)%7= 4 |
| 35 | 0 | (350%101)%7=5 |
| 51 | 2 | (510 % 101) % 7 = 5 (When a<b, a%b=a). Therefore it’s 5 |
| 42 | 0 | (420 % 101) % 7 = 2 |
| 64 | 1 | (640 % 101) % 7 = 6 |
| 71 | 1 | (710 % 101) % 7 = 3 |

|  |  |
| --- | --- |
| 0 | ~~35~~ ~~42~~ 35 |
| 1 | ~~64~~ ~~71~~ 64 |
| 2 | ~~72~~ ~~37~~ ~~51~~ ~~72~~ 42 |
| 3 | 71 |
| 4 | 37 |
| 5 | ~~35~~ 51 |
| 6 | ~~72~~ ~~64~~ 72 |

**god thats painful.**



Shahne: a hashing function is acceptable if and only if

∑ni2 ≤ 2n (ie the sum of all occupied indices squared)

In the function h(10, 0) with the keys above, ∑ni2 = 0 + 0 + 12 + 12 + 12 + 22 + 22 = 11.

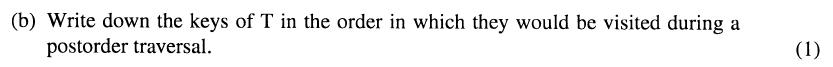
2\*n = 2\*7. 11 ≤ 14 so this is an acceptable function. I think. Shahne: it’s the table from question ii.

Kadin: If anyone is wondering what she is using for the above squared values it is simply representing positions 0 - 6 in the hash table, and how many numbers are in each, e.g if 2 numbers at position 1 and none elsewhere equation would be = 2^2 + 0 + 0 + 0 + 0 + 0 + 0

**L: Can someone else give another explanation?**

### 5. Trees

## 



31 42 53 64 75





Case 1 (red uncle: 75 becomes black)

Case 3 (outer child: bringing middle key up) were used.

Case 3 first then case 1, just incase read wrong



**BST delete****then case 4**

After BST deletion, the problem node is 75’s right null node; it had a black sibling (42) which had a red outer child (31), so case 4.  
Fixed by: make sibling (42) red; make siblings outer child (31) black; rotate sibling (42) up; make root (75) black again. and this “make (42) black again”

**It’s case 4 first: z=64, y=75, x=Y’s successor, w=42 for all those that use the rules/cases.**

**Then case 1 sincasde 53 is w’s sibling and is red, so you swap 42 and 75 colour and the extra black cancels out the 75 red leaving a balanced tree.**

Yep, look at the children of *problem-nodes-sibling*, so look at 42’s children, which are both **red,** so it’s case 4

if (Sibling == black && OuterChild == red) then {

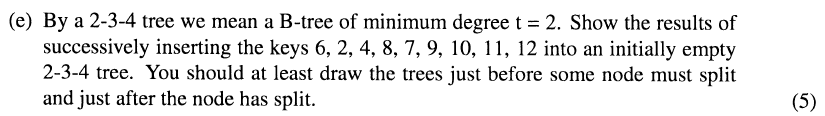
ColSwap(OuterChild, Sibling);

Rotate(Sibling, up);

}

Phil: We look at the problem node, which is a dummy leaf with 2 blacks. We looks at that node’s sibling. The sibling is black so we look at it’s children. The sibling has a red outer child. So it is case 4.

Note: there was some confusion between case 1 and 4, which is why there are several explanations







I can’t be bothered drawing circles



Do we need to use the one pass strategy? It IS an improvement, but he never really explained it much, the lecture notes kinda glaze over it.

One pass strategy isn’t necessary



They come back together before deletion occurs to make a stronger node.

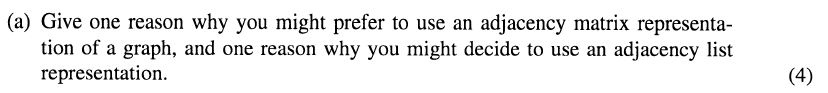
Delete 8

### 

Delete 9

### 

### 6. Graphs



The 3 differences from the lecture notes.

**1.** The adjacency list representation is more dynamic.

**2.** The adjacency matrix takes up a lot of space - if the graph is dense this space is utilised, if the graph is sparse an adjacency list would be more economical.

**3.** Sometimes we need to ask if is A connected to B

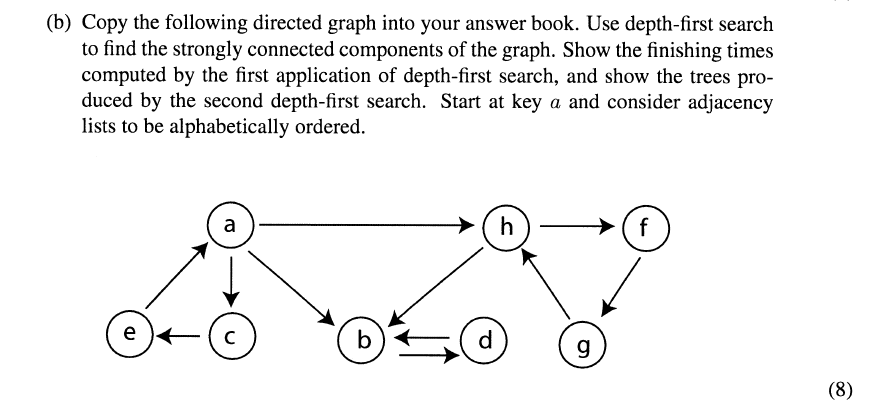
Searching the adjacency list requires O( n ) because it’s a linked list.

Searching the adjacency matrix can be done in O( 1 ), so the adjacency list is faster. *you mean matrix?*

what are these adjacency things?

An adjacency matrix is an n x n Boolean array that has a 1 represent if the two vertices share an edge, 0 if they do not. In a directed graph, 1 represents if the edge directs to the vertex, 0 if not. An adjacency matrix is useful if the graph is dense (meaning the number of edges is close to n2) and determines whether there is an edge between two vertices.

An adjacency list is a representation of using arrays that are linked to connect all vertices. An adjacency list would be used to find all vertices adjacent to a source vertex and be able to traverse a graph based on a source vertex. An adjacency list is useful is the graph is sparse (meaning the number of edges is n or fewer).



*Photo of my answer.* [*http://i.imgur.com/RzcGcWx.jpg*](http://i.imgur.com/RzcGcWx.jpg) *Didn’t need finishing times on 2nd graph.*

you’re missing the edge from h to b\*

In the transpose graph why do you go from a to h instead of a to b which is alphabetical?

Phil: I think I can answer this. In the transpose graph, once you finish a depth-first search on one tree and need to restart on another node (such as after finding A’s tree) you need to prioritize the node which had the **highest** finishing time in the **first** depth first search. B’s finishing time is 9, while H’s finishing time is 15, so H is the next node to perform DFS on.

The connect components I got for the first DFS are a(16)-b(5)-d(4); c(9)-e(8); dh(15)-f(14)-g(13) then the second one from transpose a-e-c; b-d-h-g; f

I got this; where did i go wrong? <http://i.imgur.com/unZ8adR.jpg>

First you perform DFS on the graph from ***a***, so as to find the final finishing times.

Then perform DFS on the transpose starting from the node with the largest finishing time; find your first **S**trongly **C**onnected **C**omponent; then DFS the next largest finishing time node.

### 

Dijsktra’s finds distance from source node to destination node. Give every node infinity value and cross off when distance found.

Dijkstra’s uses a min priority queue, ie the lowest cumulative weight is the highest priority.

when the algorithm starts it puts in the source ( a ), with distance 0.

and as you visit neighbours you add them to the priority queue with their respective cumulative weights. (or you could add them all with dis = inf, then just update them to make them smaller)

VERTEX PARENT

|  |  |
| --- | --- |
| a | - |
| b | a |
| d | f |
| e | d |
| f | a |

This one is the right answer for parent and vertex

I asked Willem about this and this is the correct answer.

Also the PQ : a - b - c - f - d- e

dosent matter if it looks like : a - b - f - c - d - e

So starting from a. (no parent)

a-b shortest path is : 3 (parent = a)

a-f shortest path is : 7 (parent = a)

a-c shortest path is : 7 (parent = b)

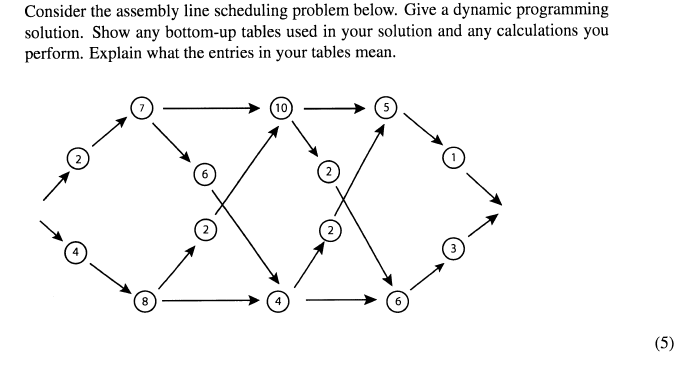
a-d shortest path is : 8 (parent = f)

a-e shortest path is : 10 (parent = d)

<http://www.youtube.com/watch?v=fCVHpnixj88> ← : - D

Shahne: I got f as the parent of d and a as the parent of f… !Forgot to cumulatively add value

**7. Dynamic Programming**

ans:

f1[j]

f2[j]

1 2 3

|  |  |  |
| --- | --- | --- |
| 9 | 19 | 23 |
| 12 | 16 | 22 |

after putting all the elements to the table we check that

if f1[n] + x15 <= f2[n] + x2

set f\* = f1[n] + x1

else f\* = f2[n] + x2

So in this case: f1[n] = f1[3] = 23 ; x1 = 1 ; f2[n] = f2[3] = 22 ;x2 = 3

if 23+1 <= 22 +3

f\* = f1[n]+x1

f\* = 24

f\* = 24 (distance of fastest route)

Each column is the station that is represented by f#[j]. f1 is the stations on top, f2 is the stations on bottom. The numbers in the columns are the shortest distance to the station.

l1[j]

l2[j]

1 2 3

|  |  |  |
| --- | --- | --- |
| we dont need to find first one | 1 | 2 |
| (s00ame as above) | 2 | 2 |

l\* = 1

l#[j] is 0the line number to the station, 1 being the top line, 2 being the bottom.

### Phil: (FIXED :D)I think l\* should be 1. l\* represents the end of the line, so which line do we come OFF from at the end of the scheduling. When 24 (f\*) is calculated, l\* is also calculated. So the way that is decided for the final addition, is what l\* should be. (Sorry if that made completely no sense, I’ll clarify myself more if asked).

^^all good i’ll fix

### 

### 8. P and NP

Note: for Aunt Maud’s sake; “Solvable in polynomial time” means that the problem can be solved in a reasonable amount of time. **LOL. :->**

P is the class of all problems that are solvable by algorithms which have a polynomial time complexity, ( O(nk) st k is a non-negative int ). Examples are sorting, hashing and long division.

NP is the class of all solvable problems whose solution can be verified in polynomial time. NP contains everything in P and NP-complete classes.

NP-Complete is a set of problems in NP such that any problem in the class can break down to the same problem as any other problem in the set. It cannot be solved in polynomial time in any known way, unless it is shown that P = NP.And that person wins so many prizes and shi….

†

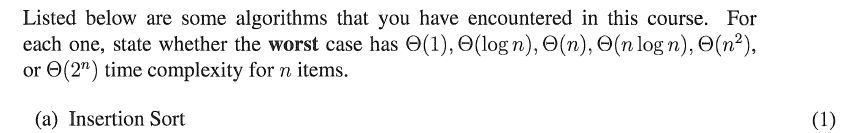
An example of an NP-complete problem is the Hamilton cycle (HCP) which is a sequence of adjacent vertices and distinct edges in which every vertex of the graph appears exactly once, except the first and last vertices are the same.



Phil: The travelling salesman problem is also NP-Complete. (Me being pedantic). Also an important quality of problems in NP is that they can be **verified** in polynomial (n^k) time.

## 2011

### 1. Complexity Classes



O(n2)



O(n log n)

|  |
| --- |
|  |
| 3(Because when there is another collision the algorithm of hashing will increase like (primary+secondary\*3)%sizeOfTable  (primary+secondary\*4)%sizeOfTable  .  .  Until it founds the empty space |



O(n)



O(n)



O(log n)



O(1)



O(log n)



2n

**2011 EXAM QUESTION 2: Recurrences**

Use the iteration method to solve the recurrence

f(1) = 5

f(n) = f(n - 1) + 2

You do not need to prove that your solution is correct.

f(n) = f(n-1) + 2 = 5 + 2 = 7

f(n) = f(n-2) + 4 = 7 + 2 = 9

f(n) = f(n-3) + 6 = 9 + 2 = 11

Therefore,

f(n) = f(n-k) + 2k

f(n) = f(n-(n-1)) + 2(n-1) = f(1) + 2n - 2 = 3+2n

This answer is insufficient ..

everyone seems to be saying that

**2011 EXAM QUESTION 3: Big-O and Induction**

Using the definition of big-O and induction, prove that n^3 = O(2^n).

n^3 = O(2^n) means for some c and n0, n^3 =< c.2^n for all cases of n>= n0

lets try c = 1

**atom case**

n=10 → 1000 =< 1024 true

**recursive case**

Assume that k is an element in Y, is k+1 an element of Y?

Induction hypothesis: k3 < 2k

if k3=< 2k then (k+1)3 =< 2k+1

We want the left hand side of the equation to match the right hand side, so we work with the left hand side.

(k+1)3 =< k3 + 3k2+ 3k + 1

We want to make 3k+1 into like terms and add: k3 + 3k2 + 4k

Add like terms for 3k2 + 4k: k3 + 4k2

Since we can assume k is at least 10, we can say 4k^2 <k^3 < 2^k due to the induction hypothesis: k3 + k3

Because k3 < 2k, change both k3 to 2k.

2k + 2k can be written as 2(k+1) which is the same as the right hand side, proving that n3 < O(2n)

**2011 EXAM QUESTION 4: Sorting**

A) Carefully describe the differences between Merge Sort and Quicksort *(5 marks)*

Merge sort has the list divided into equal parts until the parts reaches a certain size (depending on the computing machine) where another sort method (insertion or selection) is used, before merging the divided parts together in a new list.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 5 | 8 | 1 | 12 | 7 | 16 | 2 |

Divide into two equal parts

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | 5 | 8 | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| 12 | 7 | 16 | 2 |

Use sort method to sort each part (you can continue to split til you get single blocks)

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 5 | 6 | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 7 | 12 | 16 |

Using a new list, compare the first values of each list (1 vs 2, 5 vs 2, 5 vs. 7, 6 vs. 7, etc.) and add to the new list from smallest to largest.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 5 | 6 | 7 | 8 | 12 | 16 |

Quicksort has a pivot chosen from the list (which is usually the first, middle, or last key) and determines the position of the keys based on if they are greater than the pivot or less than/equal to the pivot. Once the list is done, it is split and both sides keep picking pivots and switching until all are sorted. After splitting the sublists are merged back together.

Let’s make 8 is pivot

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 9 | 8 | 3 | 12 |

i j

Move position i up the list until it reaches a key that is greater than 8, and move position j down the list until it reaches a key less than or equal to 8.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 9 | 8 | 3 | 12 |

i j

Swap the keys in these positions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 3 | 8 | 9 | 12 |

less than and equal pivot / greater than pivot

Split into two lists and continue choosing pivots and switching.

Let’s make 5 pivot

|  |  |  |
| --- | --- | --- |
| 5 | 3 | 8 |

i j

|  |  |  |
| --- | --- | --- |
| 5 | 3 | 8 |

i j

Swap values in i and j with each other

|  |  |  |
| --- | --- | --- |
| 3 | 5 | 8 |

You would continue dividing and swapping on all parts until sorted, then bring back together.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 5 | 8 | 9 | 12 |

B) Mention one way to improve Merge Sort *(2 marks)*

Use selection or insertion sort on the sub-lists when they reach a certain size. This reduces the amount of splitting and copying to be done with the new list.

C) Describe one way to improve the partitioning in Quicksort *(2 marks)*

Improve the pivot selection  
Median of three: guess your pivot value by getting the median of the *first,**middle* and *last* items. this means your pivot is more likely to be the median making the quicksort more likely to be O(n.log(n))

D) If an algorithm that sorts items by comparing key values cannot do better than O(n log n), how is it possible that one can sometimes sort keys in O(n) time? *(3 marks)*

Counting sort; it doesn’t compare key values.

counting just iterates over the unsorted array of length once so O(n). then iterates over the frequency array, which is length k (keyspace size) so O(k)

Overall time complexity is O(n + k), if k is the same size as n; then it’s O(n + n) = O(2n) = O(n)

E) What is the difference between a stable sort and an unstable sort? Mention one situation in which you really need a stable sort *(3 marks)*

Stable sorting algorithms consider not only the keys of the data but the order of items and additional information. An example would be in radix sort where sorting depends on the exact order of keys from earlier sorts.

**2011 EXAM QUESTION 5: Hash Tables**

A) Given a table of size 7, a hash function h(k), and input keys 27, 47, 81, 74, 11, 50 and 64 (in that order), draw the hash table that results from:

(i) Chaining, with h(k) = k%7.

|  |  |
| --- | --- |
| 0 |  |
| 1 | [50, 64] |
| 2 |  |
| 3 |  |
| 4 | [81, 74, 11] |
| 5 | [47] |
| 6 | [27] |

(ii) Open addressing with double hashing. Use h(k) = k%7 as the primary hash function, and   
g(k) = 1 + (k%6) as the secondary hash function.

|  |  |  |
| --- | --- | --- |
| k | k%7 | 1 + (k%6) |
| 27 | 6 | 4 |
| 47 | 5 | 6 |
| 81 | 4 | 4 |
| 74 | 4 | 3 |
| 11 | 4 | 6 |
| 50 | 1 | 3 |
| 64 | 1 | 5 |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Phil: Just expanding on what was written above. In order to change the size of a hash table, you have to rehash EVERYTHING. This is not a cheap operation (still O(n) but with a pretty large constant, I can imagine). Meanwhile, if we have a BST, we still get relatively good search, when it’s balance (O(logn)) and the size of the tree can grow or shrink much faster.

**2011 EXAM QUESTION 7: Red-Black Trees**

A) Show all the red-black trees that result after successively inserting the keys 5, 4, 3, 2, 1 into an initially empty red-black tree. State which cases apply.

case 3 uncle black x is red outer child Case 1 after adding 2? Case 3 after adding 1? yes

4(b)

2(b) 5(b)

1(r) 3(r)

B) Show all the red-black trees that result from the deletion of 4. State which cases apply.  
I think we get case 4 here

5(b)

2(b) .(b + b)

1(r) 3(r)

Case 4 (sibling has red outer child, so color swap sibling and parent, rotate sibling upwards)

2(b)

1(b) 5(b)

3(r)

**2011 EXAM QUESTION 8: B-Trees**

A) By a 2-3-4 tree we mean a B-tree of minimum degree t=2. Show the results of successively inserting the keys 7, 3, 4, 9, 8, 11, 14, 16, 18 into an initially empty 2-3-4 tree. You should at least draw the trees just before some node must split and just after the node has split.

3,4,7 -> 4 -> 4 -> 4,8 -> 4,8 ->

3 7,9 3 7,8,9 3 7 9,11 3 7 9,11,14

4,8,11 -> 8

3 7 9 14,16,18 4 11

3 7 9 14,16,18

Is the last stage here necessary? Wouldn’t it only split if there was another key inserted?

got to be prepared… its the one pass strat or something  
I think you might be confused between the one pass strat for insert and delete?

B) Show what happens when you delete key 9.

**4, 8, 14**

**3 7 11 16,18**

**2011 EXAM QUESTION 9: Graphs**

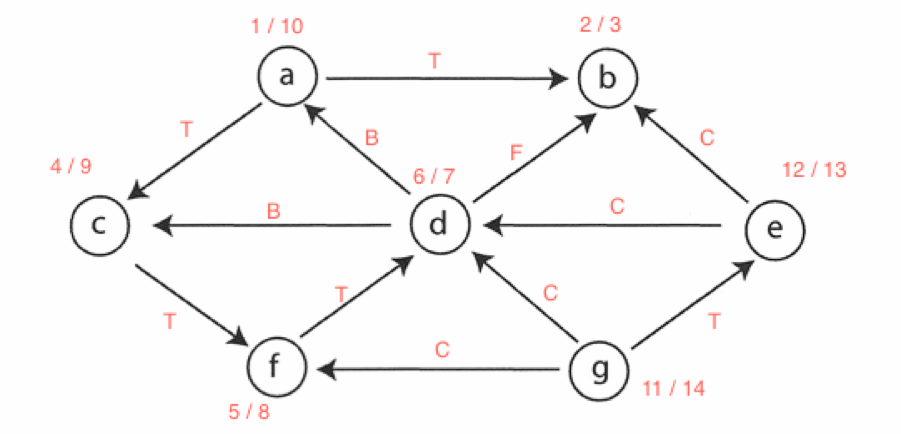
A) Copy the following directed graph into your answer book. Do a depth-first search starting at vertex a, showing time stamps, and label the edges with T, F, B, OR C according to whether each is a tree edge, forward edge, back edge or cross edge. (See 2011 exam for diagram.)

B: Just made the diagram; see here:



Having said that, I don’t think we quite covered this material?

(We covered this in lecture 19)



I have a question: why do we go to g before e? shouldn’t we go in alphabetical order when we choose the next source?

In this graph there is only one error.

if we see the relationship between d and b

The example above shows F.

But the correct answer is C

because if grey -> black

it can be F or C

if the direction is like a->b

if a>b

its C = cross edge

else

F = forward edge

d->b

Therefore d discovery time is 6 and b discovery time is 2

d>b

6>2?

Yes, so its C, cross edge

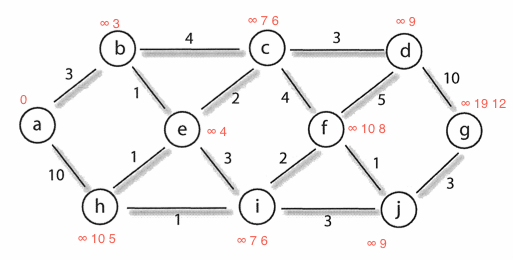
Phil: That forward edge (between d and b) should be a cross edge. A forward edge is an edge which travels to a descendent. For example:

 The black edges are tree edges. The red line is the forward edge.

Now this is a cross edge:

 The black edges are tree edges, and the green edge is a cross edge.

B) Show how Dijkstra’s algorithm works, with **a** as source. Show clearly how the priority values change, and the order in which vertices are extracted from the priority queue.(See 2011 exam for diagram.) B: But you’ve put the diagram just below, only with your working on it.



**2011 EXAM QUESTION 10: Dynamic Programming**

Consider the assembly line scheduling problem below. Give a dynamic programming solution. Show any bottom-up tables used in your solution and any calculations you perform. Explain what the entries in your tables mean. (See 2011 exam for diagram.)

Phil: Rawr, I am assuming people know how to do this now and just want answers to compare with (If you need to know how to do this question, look at 2012). Please ask if something seems wrong or needs explaining.

|  |  |  |  |
| --- | --- | --- | --- |
| j | 1 | 2 | 3 |
| f1j | 9 | 18 | 23 |
| f2j | 12 | 16 | 22 |

f\* = 24

|  |  |  |
| --- | --- | --- |
| j | 2 | 3 |
| l1j | 1 | 1 (can also be 2) |
| l2j | 2 | 2 |

l\* = 1

**2011 EXAM QUESTION 11: P and NP**

In a few well-chosen sentences, tell Aunt Maud what the classes P and NP are, and what it means to say that a problem is NP-complete. Give her one example of an NP-complete problem.

A duck walked up to a lemonade stand, and he said to the man running the stand.

Hey, bum bum bum, got any grapes? Whut?

## 

## 

## 2010

**2010 EXAM QUESTION 1: Complexity Classes**

Listed below are some algorithms that you have encountered in this course. For each one, state whether the worst case has O(1), O(log n), O(n), O(n log n), O(n^2), or O(2^n) time complexity for n items.

A) Quick Sort

**O(n log n) - Average case**

**Isn’t it O(n^2)? Yes. Worst case, which is rare (usually if the pivot is the first or the last)**

**ohhh maybe**

B) Merge Sort

**O(n log n)**

C) Searching an unordered list

**O(n)**

D) Binary Search

**O(log n)**

E) Searching a Perfect Hash Table

**O(1)**

F) Searching a Binary Search Tree

**O(n)**

**O(log n)? - Average case. A O(n) is right for worst. Binary search is O(log n), BST search is O(n)**

**worst case is an bst where they all goo to the same side, like an array**

G) Searching a Red-Black Tree

O(log n)

**2010 EXAM QUESTION 2: Recurrences**

Show by the iteration method that the recurrence

f(1) = 1

f(n) = n \* f (n - 1)

defines the function “n factorial” (i.e. n!). You do not need to prove the solution to be correct.

**2010 EXAM QUESTION 3: Big-O and Induction**

Using induction, prove that n^2 = O(n!)

n^2 = O(n!)

means n^2 < c.n! for all n>n0

try c = 1

n^2 < n!

n=1 1 < 1 yes

n=2 4 < 2 no

n=3 9 < 6 no

n=4 16<24 yes

atom case

n = 4

n^2 < n! induction hypothesis

16 < 24

closure case

n^2 < n!

seeing k^2 < k! works (when n0 = 4) does (k+1)^2 < (k+1)!

LHS = (k+1)^2

= k^2+2k+1

< k^2+3k (because 1<k0(=4))

< k^2+k^2 (because 3<k0(=4))

< k!+k! (k^2 < k! (induction hypothesis))

< (k+1)k!

< (k+1)!

< RHS

closure case is prove QED

**2010 EXAM QUESTION 4: Sorting**

A) Carefully describe the differences between Merge Sort and Quicksort.

Merge sort divides the array in half by equal parts until they are single blocks, then compares each block to the ones next to it and sorts and merges the lists together appropriately. I got a picture up in 2011 that’s a lot better.

Quicksort divides the array using a random pivot to determine numbers that are greater and less than it’s value, then continues to ‘divide and conquer’ until they are all set.

B) Suppose you are to implement a sorting algorithm for a search engine. It will sort all of the words in all of the documents in a very large collection: this will take quite a long time and memory will be tight. What sorting algorithm would you implement, and what would you do to make it perform as well as possible?

C) Draw the bucketing structure produced by Bucket Sort when sorting the following data:

0.34, 0.54, 0.88, 0.81, 0.11, 0.50, 0.71

|  |  |
| --- | --- |
| 0 |  |
| 1 | .11 |
| 2 |  |
| 3 | .34 |
| 4 |  |
| 5 | .54 - .50 |
| 6 |  |
| 7 | .71 |
| 8 | .88 - .81 |
| 9 |  |

D) What is the difference between a stable sort and an unstable sort? Describe one situation in which a stable sort is necessary.

A stable sort depends on additional information such as the order of items, rather than just the keys in the data comparing each other. An example of a stable sort would be radix sort where earlier sorts affect the outcome of the later sort.

**2010 EXAM QUESTION 5: Hash Tables**

A) Given a table of size 7, a hash function h(k), and input keys 34, 54, 88, 81, 11, 50, and 71 (in that order), draw the hash table that results from:

(i) Chaining, with h(k) = k%7.

|  |  |
| --- | --- |
| **k** | **k%7** |
| 34 | 6 |
| 54 | 5 |
| 88 | 4 |
| 81 | 4 |
| 11 | 4 |
| 50 | 1 |
| 71 | 1 |

|  |  |
| --- | --- |
| 0 |  |
| 1 | 50,71 |
| 2 |  |
| 3 |  |
| 4 | 88,81,11 |
| 5 | 54 |
| 6 | 34 |

(ii) Open addressing, with double hashing. Use h(k) = k%7 as the primary hash function, and g(k) = 1 + (k%6) as the secondary hash function.

|  |  |  |
| --- | --- | --- |
| **k** | **k%7** | **1+(k%6)** |
| 34 | 6 | 5 |
| 54 | 5 | 1 |
| 88 | 4 | 5 |
| 81 | 4 | 4 |
| 11 | 4 | 6 |
| 50 | 1 | 3 |
| 71 | 1 | 6 |

|  |  |
| --- | --- |
| 0 | 50 |
| 1 | 81 50 |
| 2 | 71 |
| 3 | 11 |
| 4 | 88 81,11 50 |
| 5 | 54 |
| 6 | 34 |

(iii) Chaining, with universal hash function h(10,10)(k) = ((10k + 10) % 101) % 7.

|  |  |  |  |
| --- | --- | --- | --- |
| **k** | **k%7** | **1+(k%6)** | **((10k+10)%101)%7** |
| 34 | 6 | 5 | 350%101 = 47%7 = 5 |
| 54 | 5 | 1 | 550%101 = 45%7 = 3 |
| 88 | 4 | 5 | 890%101 = 72%7 = 2 (//890%101 = 82%7 = 5) ty |
| 81 | 4 | 4 | 820%101 = 12%7 = 5 |
| 11 | 4 | 6 | 120%101 = 19%7 = 5 |
| 50 | 1 | 3 | 510%101 = 5%7 = 5 |
| 71 | 1 | 6 | 720%101 = 13%7 = 6 |

|  |  |
| --- | --- |
| 0 |  |
| 1 |  |
| 2 |  |
| 3 | 54 |
| 4 |  |
| 5 | 34,88, 81,11,50 |
| 6 | 71 |

B) Suppose you were using a perfect hashing scheme to create a hash table from the keys above. What would be the cost of using h(10,10) as the primary hash function? Why is that cost considered unacceptable?

cost = Σ c***i***2 where c***i***is the number of ~~collisions~~ values at index ***i***

cost = 02 + 02 + 02 + 12 + 02 + 52 + 12

cost 0+0+1+1+0+25+1 = 28

if it was acceptable 19 < 2\*n = 14 (where n is the length of the array)

28>14 :(

**not cool man**

C) Suppose you are using double hashing with the secondary hash function g as described above. Explain why a hash table of size 10, 100 or 1000 would be a poor choice, regardless of the number of keys.

**2010 EXAM QUESTION 6: Binary Search Trees**

A) Draw the final binary search tree T that results from successively inserting the keys 1, 5, 2, 4, 3 into an initially empty tree.

1

5

2

4

3

B) Write down the keys of T in the order in which they would be visited during a postorder traversal.

3-4-2-5-1

C) Draw the results of deleting 1, then 5, then 2 from T.

5 2 4

2 4 3

4 3

3

D) Give one reason why you might choose to store data in a binary search tree instead of a hash table.

Dynamic size. dont have to set size at the start. so if you dont know how big it will be

to show relationships

**2010 EXAM QUESTION 7: Red-Black Trees**

A) Show 1, 5, 2, 4, 3 into an initially empty red-black tree.

1 -> 1 -> 1 -> 2 -> 2 -> 2 -> 2

5 5 1 5 1 5 1 5 1 4

2 4 4 3 5

3

B) Show all the red-black trees that result from the successive deletion of 1, then 5, then 2.

2 -> 4 -> 4 -> 3 -> 3

1 4 2 5 2 2 4 4

3 5 3 3

Why not? 3 not too sure

2 4

Phil: Because of the way we handle case 4 in deletion. After deleting 5, we get a case 3 problem. So we change it into case 4 by swapping the colours of 2 and 3 and performing a left rotation. We then have and must perform a right rotation, and then push the +b from the dummy node up to the root, resulting in the entire tree being black.



**2010 EXAM QUESTION 8: B-Trees**

A) By a 2-3-4 tree we mean a B-tree of minimum degree t=2. Show the results of successively inserting the keys 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 into an initially empty 2-3-4 tree. You should at least draw the trees just before some node must split and just after the node has split.

8,9,10 -> 9 -> 9 -> 7,9 -> 7,9 -> 5,7,9 ->

7,8 10 6,7,8 10 5,6 8 10 4,5,6 8 10 3,4 6 8 10

7 -> **7**

5 9 **3, 5 9**

2,3,4 6 8 10  **1,2 4 6 8 10**

B) Show what happens when you delete first key 4 and then key 1.

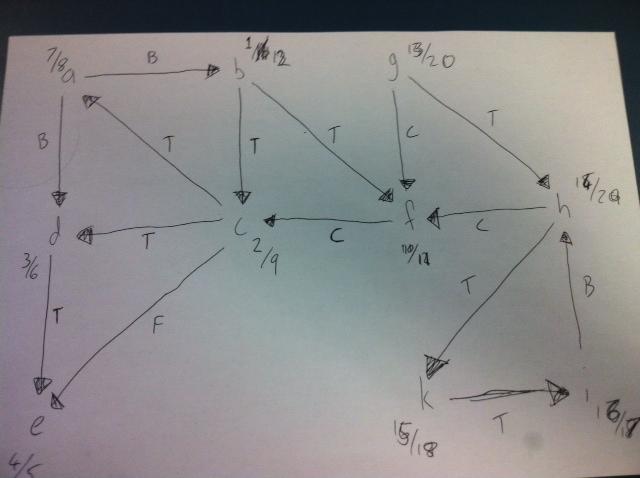
7 -> 7 -> **7**

3, 5 9 2, 5 9  **5 9**

1,2 4 6 8 10 1 3 6 8 10  **2,3 6 8 10**

**2010 EXAM QUESTION 9: Graphs**

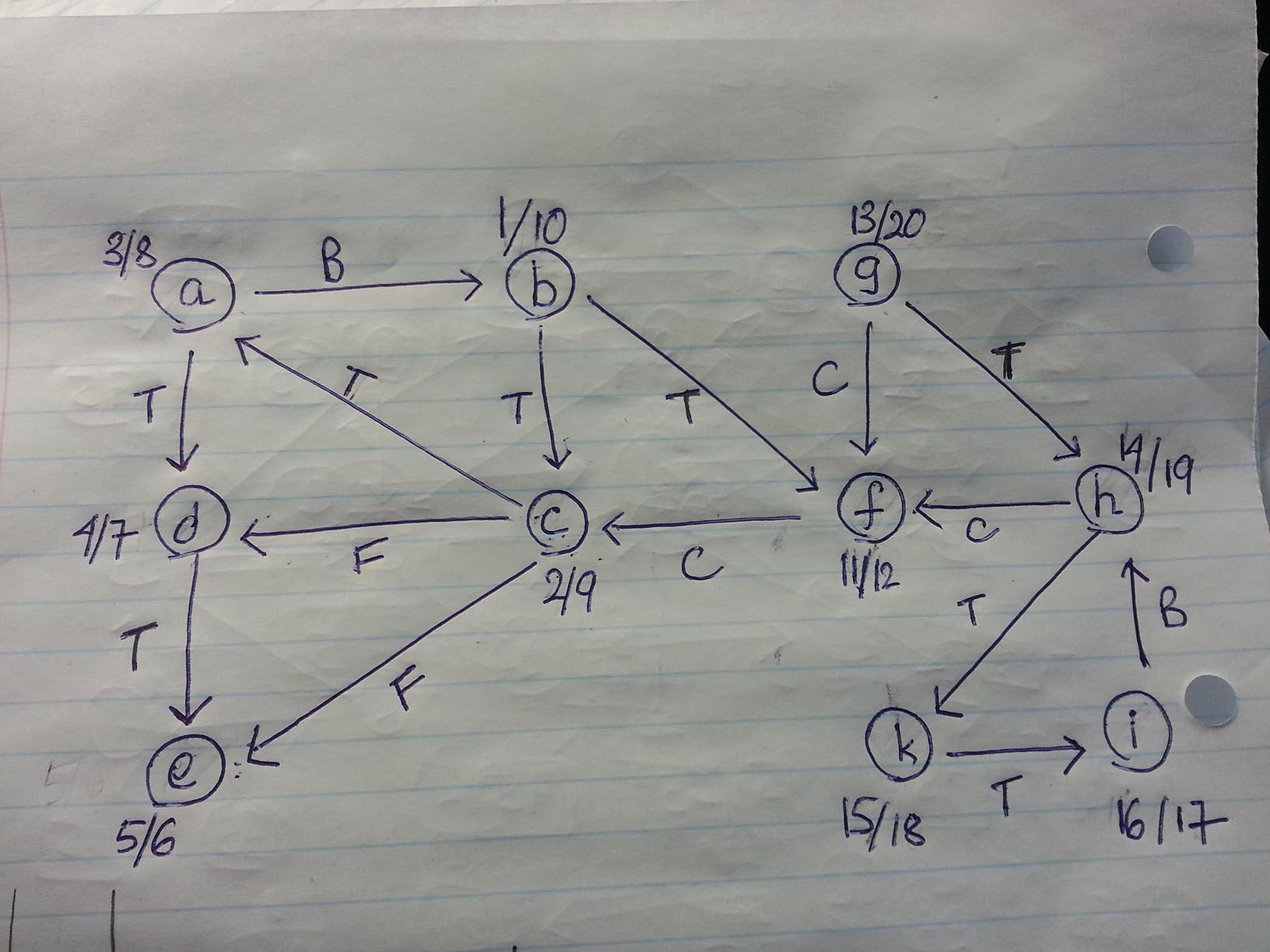
A) Copy the following directed acyclic graph into your answer book. Do a depth-first search starting at vertex b, showing time stamps, and label the edges with T, F, B, or C according to whether each is a tree edge, forward edge, back edge, or cross edge. (See 2010 exam for diagram).



Aung - Why does ‘a’ become ⅞ instead of ⅜? There are 3 ways to go from C: a,d or e. Shouldn’t we choose ‘a’ according to alphabetic order?

Here is my answer.

**i thought it was closest value of current node**



B) Copy the following directed graph into your answer book. Show how Dijkstra’s algorithm works, with a as source. Show clearly how the priority values change, and the order in which vertices are extracted from the priority queue (See 2011 exam for diagram).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **vertex** | **parent** |  | **vertex** | **distance** |
| a | - |  | a | 0 |
| d | a |  | b | 2 |
| b | a |  | c | 3 |
| c | d |  | d | 1 |
| e | d |  | e | 3 |
| g | d |  | f | 5 |
| f | g |  | g | 6 |

PQ: a b d c e f g

**2010 EXAM QUESTION 10: Dynamic Programming**

Consider the assembly line scheduling problem below. Give a dynamic programming solution. Show any bottom-up tables used in your solution and any calculations you perform. Explain what the entries in your tables mean (See 2011 exam for diagram).

staions

|  |  |
| --- | --- |
| **station 1** | **station 2** |
| 7 | 7 |
| 5 | 11 |

f\* = 12

**l\* = 2 use bottom path**

**2010 EXAM QUESTION 11: P and NP**

In a few well-chosen sentences, tell Aunt Maud what the classes P and NP are, and what it means to say that a problem is NP-complete. Give her one example of an NP-complete problem.

Phil: NP-complete means that any problem in NP can be reduced to an NP-complete problem. The travelling salesman problem is an example of an NP-complete problem. You must travel to every node with the total cost being under some threshold value. This can be verified in P time, if the routes cost is less than the threshold value, then you know the answer you have is legit.