**COSC242 EXAM PREPARATION**

Hey guys, please feel free to jump in and contribute your vast quantities of COSC242 wisdom.

If you are adding exam questions, please pop them under the correct heading, so we can all keep track of what’s happening.

Also feel free to comment below with any questions or topics you would particularly like covered in the review lecture, and we can pass this on to the lecturer beforehand.

The more people on board, the better this works, so spread the word.

Comment below with your name and chosen colour, so it’s clear who is contributing what (this will be the colour you use when answering past exam questions).

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\*\*\*\* old names \*\*\*\***

Charlie

Andy

Elliot

Codie! ← Awesome name for a CompSci student Thanks...? :D

Tim

**Regan**

Jake

Tim2

Stuart

Nick

**Abs**

**Mellor**

Louis

Alan Statham.

Campbell

**Ben**

**Lewis**

**Juni**

***campbell2***

***Jenny***

**JD**

**Ben H**

***Corwin***

***Cameron***

**\*\*\*\* old names above here \*\*\*\***

**2013 people:**

***Scott vH***

I made a new doc

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Questions/Topics for Review:**

* **It seems reasonably likely we’ll get the same induction question as we did in the assignment, going by 2011. They had prove n^3 = O(2^n) in the assignment and then the same in the final exam below.** What an Eagle eyed viewer.

**Does anyone have a copy of how we did the induction for the assignment?**

***Yes, it’s at the end of the document.***

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Sections and marks allocated (Can someone complete this?) :D Cheers!**

**1) Complexity(8)**

**2) Recurrences, Big O and Induction (10)**

**3) Sorting (10)**

**4) Hash tables (22)**

**5) Trees: Binary Search Trees, Red Black Trees, B Trees (20)**

**6) Graphs (20)**

**7) Dynamic Programming (5)**

**8) P and NP (5)  
Total = 100**

***Who changed the marks? These are incorrect. The complexity is 8 questions worth 1 mark each, and the second section is worth 10.***

**2011 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 ***Θ***(1), ***Θ***(log n), ***Θ***(n), ***Θ***(n log n), ***Θ***(n^2), or ***Θ***(2^n) time complexity for n items.

***For all of these, it is Θ not O. All the functions are O(2^n) but only one is Θ(2^n). Should be in our answers too.***

A) Insertion Sort

O(n^2)

B) Merge Sort

O(n log n)

C) Counting Sort

O(n)

D) Searching an unsorted array

O(n)

E) Binary search

O(log n)

F) Searching a Perfect Hash Table

O(1)

G) Searching a Red-Black Tree

O(log n)

H) Finding all the subsets of a set

2^n

**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)

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

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

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

so f(n) = ((f(n - 3) + 2) + 2) + 2 = f(n - 3) + 6

At this point we can note that the pattern emerging is:

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

We need to get rid of 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)) + 2(n - 1) = f(1) + 2n - 2 = 2n + 3

So f(n) = 2n + 3

It would be good to check f(1) = 2 \* 1 + 3 = 5

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

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

**Show that there exists c(constant), n0 such that n^3 < c\*2^n for all n > n0**

**(Atom)**

**When n = 10 n^3 = 1000 c\*2^n = 1024**

**So take n0 = 10 and c = 1**

**Let Y be the set of all n such that n > 10 and n^3 < 2^n**

**k**

**(Closure)**

**Let k є Y (induction hypothesis so pick any random element out of Y)**

**We know that k3 < 2k so k will be at least 10**

**Is k + 1 є Y?**

**Note: to 242 peps this is a very small step by step and yellow highlight where there is “animal” change.**

**(k + 1)^3 = (k + 1)(k + 1)(k + 1)**

**= (k + 1)(k^2 + 2k +1)**

**= k^3 + 2k^2 + k + k^2 + 2k + 1**

**= k^3 + 3k^2 + 3k + 1**

**<k^3 + 3k^2 + 3k + k (because k is at least 10: n0 = k = 10 >= 1)**

**= k^3 + 3k^2 + 4k**

**< k^3 + 3k^2 + k^2 (because k^2 is always >= 4k where k = 10 from hyp**

**so the inequality holds)**

**= k^3 + 4k^2**

**< k^3 + k^3 (same idea, k^3 is always bigger than 4k^2 where k = 10)**

**= 2\*k^3**

**< 2 \* 2^k (ind. hyp.)**

**= 2^(k+1)**

**So k + 1 є Y**

**So Y = { n |(Such that) n > 10}.**

**Just added a few more notes. Great answer.**

**2011 EXAM QUESTION 4: Sorting**

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

Quicksort is almost always faster except in the absolute worst case scenarios for quicksort or when operating on a linked list which has very slow random access performance. Mergesort tends to be more processing intensive and quicksort is more storage intensive except for linked lists. Quicksort's speed will be degraded if it does not have fast random access to the data. For example any array based sorting will be much faster with quicksort.

Quicksort has a worst case runtime of O(n^2), where Mergesort has a worst case runtime of O(n log n). This would seem to indicate that mergesort is faster than quicksort but in reality this turns out not to be the case. Good implementations of quicksort have an average runtime that is O(n log n), the same as mergesort. In addition the coeffiecient, n, for quicksort on average is much smaller than the coefficient for mergesort. Thus in practice quicksort is almost always faster than mergesort. Linked lists are anu exception to this because the storage requirements for mergesort are much lower O(1) compared to quicksort Ω(n).

Both take a divide-and-conquer approach to sorting and both are implemented recursively by splitting the list to be sorted into smaller and smaller pieces.

MS works by recursively splitting the list into two equal parts until it is reduced to single blocks and then merging the blocks back up in sorted order.

QS works by recursively dividing the list into two parts around a pivot value and then merging the sub-lists.

MS guarantees that each split is at the mid-point and produces sub-lists of equal size, it can’t guarantee the split at the mid-point if the list’s length is an odd number, one side may be + 1 bigger.. lol

In QS there is no control over where the split occurs as the pivot value is chosen randomly and is unlikely to produce two sub-lists of equal size.

In MS the splitting-point is fixed - it is the mid-point - but in QS different options can be used for determining the pivot, eg take the first value in the list or the median of the first, last and mid.

The major downside with MS is the fact it requires extra storage space for an additional list during the merge operation.

QS does not require any additional memory - it is an in-place sorting method.

Both QS and MS can be O(n log(base 2) n) - but QS can degenerate to O(n^2) if the worst case pivot is chosen each time the list is divided (for example if you use the first value each time and the list is already sorted).

To improve performance even further in MS change to Insertion Sort once the list has been been broken down to a size of 20 items.

In QS to improve performance stop dividing the list once the size gets down to 10 items and when the call to QS has returned the almost sorted list run Insertion Sort on it (insertion sort being very efficient on almost sorted data).

***The first answer says quicksort is always faster than mergesort, but this isn’t true. The worst case of mergesort is equivalent to quicksort’s best case in terms of comparisons.***

In class we covered the fact that quick sort was quite resistant to avoiding worst-case performance. Even when we partition at a 1:9 ratio, we still get O(n log n) performance. What is being stated is that quick sort is generally faster in real-world applications, due to the fact that the coefficient hidden in the average case is much better than the average case for merge sort (which is also best/worst). Quick sort can be much slower if it fails to partition efficiently though.

***What I was trying to get at is that the word faster is pretty ambiguous and is affected by things like cpu caching, recursion and the like. But in terms of comparisons merge sort is “faster”. So saying quicksort is faster isn’t really true and depends on a lot of external factors. From wikipedia “In the worst case, merge sort does about 39% fewer comparisons than*** [***quicksort***](http://en.wikipedia.org/wiki/Quicksort) ***does in the average case”***

That’s interesting. Basically, what we’ve learned is that quick sort is quicker, as the partitioning happens *n* times, whereas in merge sort the merg...0e happens 2*n* - 1 times (?). But an iterative implementation of merge sort would probably be quicker if it is making far less comparisons. So I would state that the hidden coefficient in O(n log n) is smaller for quick sort, but merge sort makes far fewer comparisons on average, and we also have the cost of memory allocation. So performance is quite dependent on the architecture, and whether recursive or iterative implementations are used. It should also be noted that both can be parallelised, but it is easier to parallelise merge sort.

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

Drop to insertion sort on the sub-arrays when they reach a predetermined length as insertion sort requires fewer steps for small values of n and is therefore more efficient. Research suggests the optimal change-over point may be between 40-50 items but to account for differences in implementing machines, 20 is a commonly accepted for the change.

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

The current implementation of QuickSort naively partitions on either side of a randomly chosen pivot, in the hopes that this pivot (approximately) evenly divides all of the other items in two. This can be improved by using the Median of Three Strategy, which involves selecting the first, middle and last items in our partition. We select the median of the three, which gives us a rough guess at the median of the whole partition, thus improving the likelihood of selecting a good pivot.

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)*

By knowing extra information about our items, we can sometime sort them without having to perform comparisons, thus avoiding the O(n log n) bound.

(Probably give a few examples, such as radix and counting sort? I feel like you would need more here)

O(n log n) can be bettered when you have additional information about the range of keys in the list and you can sort against an external value rather than having to compare keys with each other. This is the basic premise in bucketing sorting which is used in Counting Sort and Radix Sort which can sort in O(n). The efficiency is the result of less comparisons being made.

So, non-comparison based algorithms can insert the element into its final sorted position, giving the upper bound of 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)*

A sorting algorithm is described as stable if equal elements are in the same relative order in the sorted sequence as in the original sequence. If you care about the order of "equal" elements — for example, if you are sorting the same data more than once in order to combine comparison criteria, or if you want to be sure that the sort will complete in a reasonable amount of time, you should use a stable sort. Radix Sort is an example of where stable sorting is required as later sorting relies on the exact order of keys from earlier sorts. **You would need a stable sort if the data are not solely defined by their keys. An example is a hash table, where you have (key, value). The order in which you place the keys into the output matters, even if they are the same key, due to the difference in values.**

**A stable sort is one where the order in which the items occur is important/considered.** *Not sure on situation?* Radix Sort is an example of where stable sorting is required as later sorting relies 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.

**h(27) = 27 % 7 = 6**

**h(47) = 47 % 7 = 5**

**h(81) = 81 % 7 = 4**

**h(74) = 74 % 7 = 4**

**h(11) = 11 % 7 = 4**

**h(50) = 50 % 7 = 1**

**h(64) = 64 % 7 = 1**



(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.

The primary hash function is the same as the previous question - so only need to calculate the step using the secondary hash function for the collisions (74, 11 and 64)

**g**(74) = 1 + (74 % 6) = 3 - will end up in slot 7 (3 places beyond slot 4)

(would this end up in slot 0 since the size table is 7??

ie. index 0 - 6) **Yeah quite possible since table size was  
 wrong before.**

**g**(11) = 1 + (11 % 6) = 6 - will end up in slot 3 (6 places beyond slot 4)

**g**(64) = 1 + (64 % 6) = 5 - will end up in slot 3 (slot 2 possibly)(**~~5~~** places beyond slot 1 plus another **~~5~~**places as **~~6~~** is already taken) ? *~~What I got was it originally hashes with h(k) to 1, but it’s already taken by 50 so we use the secondary function g(k) which gives us the step 5. That leads us to position 6, which is taken by 27. So we use the step again which leads back to position 3, which is empty. So yeah, you’re right, although I did make a couple of corrections to the reasoning (probably typos)?~~***This might not be valid still since I had the table size in the diagram slightly wrong. Are you guys sure about this one. I started off finding 64 goes into slot 1, just like in question i, and then I used the g function to find that it increases in steps of 5,**

**just like you did, but from slot 1 (+step 5) to slot 6. slot 6 is occupied by 27 so again you step up 5 and go to slot 4 which has 81 so you step again by 5 and then you get slot 2, not 3. Did I go wrong somewhere? Yeah you are right, it looks like purple is counting from 1. sweet, thanks for confirming. I’ll change it at the top. Yup, not sure how that was missed haha. Nice one.**

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

**h10, 10 (27) = ((270+10) % 101) % 7**

**= (280 % 101) % 7**

**= 78 % 7**

**= 1**

**h10, 10 (47) = ((470+10) % 101) % 7**

**= (480 % 101) % 7**

**= 76 % 7**

**= 6**

**h10, 10 (81) = ((810 + 10) % 101) % 7**

**= (820 % 101) % 7**

**= 12 % 7**

**= 5**

**h10, 10 (74) = (740 + 10) % 101) % 7**

**= (750 % 101) % 7**

**= 43 % 7**

**= 1**

**h10, 10 (11) = ((110 + 10) % 101) % 7**

**= (120 % 101) % 7**

**= 19 % 7**

**= 5**

**h10, 10 (50) = ((500 + 10) % 101) % 7**

**= (510 % 101) % 7**

**= 5 % 7**

**/ = 5 (a % b where a < b is always a)**

**h10, 10 (64) = ((640 + 10) % 101) % 7**

**= (650 % 101) % 7**

**= 44 % 7**

**= 2**



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

**A perfect hashing scheme means that all keys hash to different positions in the table.**

**If , then we are happy with that function as the primary hash function.**

**We must perform the first pass step of perfect hashing, working out which slot each key will be hashed to (using h10,10).**

**We can reuse this from the previous question. We then square (the number of keys in each table row) those numbers to get ni2.**

**n02 = 0, n12 = 4, n22 = 1, n32 = 0, n42 = 0, n52 = 9, n62 = 1**

**So the sum of all those () = 15. The table size is 7 so = 14. 15 is > 14, So h10,10 is *not* an acceptable primary hash function.**

**Looks like a model model answer - nice job!**

**Sorry i have trouble understanding n02 = 0, n12 = 4, n22 = 1, n32 = 0, n42 = 0, n52 = 9, n62 = 1 here. How to get these numbers? I mean why n12 = 4, n52 = 9 ?**

**It looks like he’s squaring the number of values of each key. In pos 1 there are 2, so 22 etc, haha whoops, just saw green’s answer below. Thank u for u help as well : P**

These numbers are from the cells above in ‘a iii’ that contain numbers, n1 contains both 27 and 74 which is 2 numbers, so 22 is 4, where n1 is replaced with the amount of numbers in that location and so on, numbers that = 0 are due to no number being present. Make sense?

**Thank you so much, i get it now : P**

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.

Table size of 10, 100 or 1,000 would be a bad idea because these are factors of 10 and even numbers. An even numbered step will always land on an even numbered slot, and there will be clustering leading to lots of collisions ???

If using 10, for example, you would always get the same key value for numbers ending in a particular digit. 14%10 = 4, 24%10 = 4 and so on. Increased collision rate.

Taking the modulus of division by a prime number results in an even distribution of values. When using a value that is nonprime, such as 10, we obtain a less even distribution. This is due to the fact that some hash functions produce values that may have a common factor with the table size, resulting in an uneven distribution. The table size being prime prevents this.

Also to note, if a table is a factor of 10 or even, your stepping function only has access to a small portion of the table and can’t cycle through all cells of the table, resulting in an infinite loop. Correct me if I’m wrong.

**2011 EXAM QUESTION 6: Binary Search Trees**

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



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

**1, 2, 3, 4, 5**

C) Draw the results of deleting 4 from T.

-

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

**A BST allows us to store data in a structured way which allows us to maintain the order of data inserted into it.** *(Someone can probably write something better..)*

**The way that BSTs store values allows us to easily create lists from the stored data that can be in sorted order, preorder, postorder etc.**

**Another reason is that BSTs are more memory efficient than Hash tables. We only need to store the amount of data for which there are tree nodes, while a Hash table requires an initial table size to be specified which is generally larger than the number of items in the hash table.**

BST can be dynamically expanded as you need to add more keys while a hash table size is fixed when it is created and is therefore inflexible (i like this one, short and simple)

**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.



**Have tested this out on** [**http://gauss.ececs.uc.edu/RedBlack/redblack.html**](http://gauss.ececs.uc.edu/RedBlack/redblack.html) **and it gets the same tree, so I guess it’s correct. I’ve assumed a few steps here and there. I guess this^ is what they mean by “show all the red-black trees”?**

**ujh**

hmm...I wonder if we should show all intermediate steps in the exam, like the colourings and stuff? Couldn’t hurt

**Yeah, maybe.. the only thing I haven’t drawn a new tree for is inserting the new nodes, seemed unnecessary to draw a whole tree to just tack a node on the end. But yeah, wouldn’t hurt and might be worth it just to be clear. For the colourings, you’d probably alter the colours after you’d redrawn it, so crossing out the old and writing in the new is probably enough.**

B) Show all the red-black trees that result from the deletion of 4. State which cases apply.

**Answer makes sense, but deletion is still a mindfuck, so someone else please check it..**

Your answer looks good to me!

**That Java applet linked to in previous question doesn’t work since it picks the predecessor instead of the successor when doing the BST deletion, so it gets a different tree.**

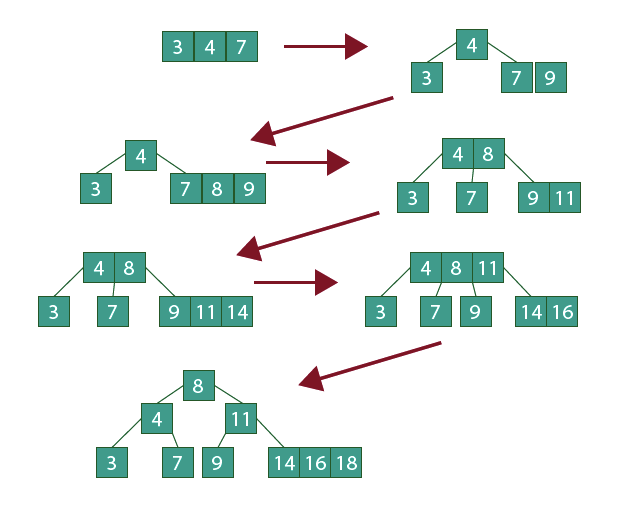
**This link uses the successor:** [**http://www.cse.yorku.ca/~aaw/Sotirios/RedBlackTree.html**](http://www.cse.yorku.ca/~aaw/Sotirios/RedBlackTree.html)

**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.

Each node can contain a minimum t-1 and maximum 2t-1 keys (i.e. in this case, between 1 and 3 keys).

Can check your answers with: http://ats.oka.nu/b-tree/b-tree.manual.html

This should hopefully be correct, if it’s not change it

Looks good to me - nice job and well done for constructing the image for us

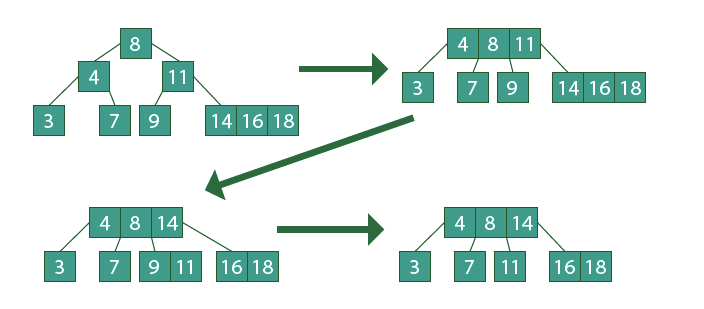
**So if on the way down we see a node which is full (ie. 4, 8, 11 node) it should be split before continuing on? -** yes, if you are inserting, if you are deleting then strengthen any inner nodes with t-1 keys

**Insertion, you come from the top down and split full nodes.**

**Deletion, you come from the top down strengthening nodes by merging/borrowing.**

B) Show what happens when you delete key 9.

Hopefully this is correct too



Looks good

**Instead of bringing down the 8 to merge with the 4 and 11, couldn’t we just borrow the 14 from the leaf node below? So we would have 11 and 14 in a node, and then we can move the 11 down to be with 9. Then we delete the 9. We end up with the same tree (except the root node is split). What do you guys think?** Makes sense, EXCEPT if you are entering to delete then you must strengthen internal nodes with only 1 key as you pass, so you merge 8, 4 and 11 even before you get to the node with the key you want to delete (as the diagram shows)

**I got what is in the diagram. Since the neighbour has a spare key we can use to strengthen, I’m pretty sure we do the pull-down-push-up thing. Wait I’m confused. You mean you did the merge like above in diagram (drop down 8), or you borrowed the neighbour (pulled up 14)? I assume you’d have to merge like the diagram, since the node with 8 has the minimum amount of keys, and has no neighbour? I could be wrong, do the notes contradict the diagram? 8 does not need strengthening, as it is the root of the tree (lecture 17 slide 5 - “Only the root may have fewer than t -1 keys).**

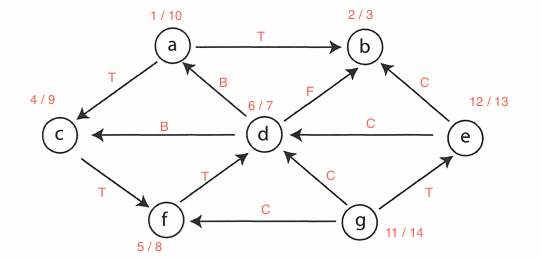
**Ah right, of course. But then you’d get to 11 and see it needs strengthening, and 4 doesn’t have a key it can borrow, so you have to merge?**

**You are right when you say 11 needs strengthening, but I assumed this was where you looked to borrow from 14 first before resorting to merge. This is where it gets confusing - both approaches seem to work in this case, but just which one to pick?**

**Yeah, I’m a bit confused too... I’ll ask Willem on Monday :) Or if someone can figure it out ,that’s cool too.  
Actually I think you are right, as 14 is not a *sibling* of 11, so I think you have to merge. I’ll still ask Willem, confusing! *I think you get it, we can only borrow keys from the sibling nodes rather than the children when strengthening an internal node.***

**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.)



**The edge from D -> B should be a cross edge since discovery time of d > b, instead of a forward edge as I’ve got there. Seems like I’ve got a lot of cross edges though, and no forwards.. so maybe I’m doing it wrong.**

**I think you are correct, I got a cross edge and no forward edges. The b is not a descendant of d, as the chain starts from a. If that makes sense :)  
Also I think g and e were visited in the wrong order, they should be the other way round, but this might not matter.**

***~~I guess alphabetical order would be a more systematic way of approaching it, rather than looking ahead and seeing which will work best. I think in an example in the notes we chose one arbitrarily which ended up having not to choose another source node (which we’d have to do since e can’t get to g).~~* Asked Willem, he said unless the question asks you to use any order, the node you choose doesn’t matter.**

**Grey - white = Tree edge**

**Grey - grey = Back edge**

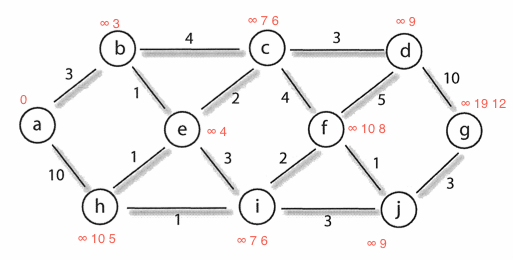
**Grey - black = (if discovery time from source to destination decreases, cross edge)**

**(if discovery time from source to destination increases, forward 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.)

HOW DO WE DO THE PRIORITY QUEUE? PLEASE HELP!!!

HOW DO WE DO THE PRIORITY QUEUE? PLEASE HELP!!!



**Order of things pulled off PQ:**

1. **a**
2. **b**
3. **e**
4. **h**
5. **c**
6. **i**
7. **f**
8. **d**
9. **j**
10. ***g (It’s removed from the PQ, just not used)***

**(Shouldnt there be a G here?) - yeah, it’s omitted because it’s implied.**

**Again, could be wrong so please check & correct if need be.**

It looks good, c and i could be visited in the opposite order as both have a value of 6 when you finish with h - but either is correct.

**^ I think G has to be explicitly removed, as some algorithms(not sure if Willem agrees) rely on the removal of the destination node in order for the algorithm to finish.**

**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.)

|  |  |  |  |
| --- | --- | --- | --- |
| *J* | *1* | *2* | *3* |
| *F1[J]* | 9 | 18 | 23 |
| *F2[J]* | 12 | 16 | 22 |

F\* = 24

|  |  |  |
| --- | --- | --- |
| *J* | *2* | *3* |
| *L1[J]* | 1 | 1 |
| *L2[J]* | 2 | 2 |

L\* = 1**, ~~should it be 2?~~ [see below] L\* is the “line number whose last station is used in the fastest way through the factory”, so by checking if F\* was calculated by using the last station in 1 or 2, and it’s 2 in this case. Not 100%, I forgot most of this stuff lol..** ~~Reading that would make it seem that way, however in the example (L23 S1) I’m sure L\* becomes the first route taken, and as per the pseudo code(L22 S11) I think it’s suppose to be the first route... correct me if I’m wrong.~~

*if f1 [n] + x1 ≤ f2 [n] + x2*

*then sef f\* = f1 [n] + x1*

*set l\* = 1*

*else set f\* = f2 [n] + x2*

*set l\* = 2*

**Yeah I think I made a mistake. Since 23 + 1 is better than 22 + 3 for the offloading at the end, you’d be coming off the assembly line from Line 1, which fits with your answer (and the english definition of what L\* is). So working backwards, the route you would get goes entirely along assembly line 1 to get from start to end.. do you get the same sort of thing?**

Correct these If I am wrong, I can’t quite recall how to do it from the lecture. As for the explain what the entries mean, i’m not entirely sure how to describe them, so any suggestions please add :D

* **F\* is the fastest time we can get an iPhone through the assembly line.**
* **L\* is the line number of the last station used in the fastest time route**
* **The values in the F*i* [J] table are the fastest times to the stations**
* **The values in the L*i*[J] table are the lines in which the fastest time to that station is accomplished**

Ahhh I get what they mean by describe haha, I’m an idiot...

**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.

**The P class contains problems which can be solved using O(nk), i.e. Polynomial Complexity. P is the set of problems that can have solutions found in polynomial time. It lies inside the NP class. The NP class contains problems whose proposed solutions can be checked by polynomial algorithms.**

**NP is a set of problems who can have a solution verified in polynomial time. A problem is NP-Complete if it is in both the NP, and NP-hard classes. It has not been proven that NP-complete problems cannot be solved with polynomial complexity, however it is considered unlikely.**

**NP Complete is a set of problems in NP such that any problem in this set can break down to the same problem as any other problem in the set. This is why if you prove one solution in NP-Complete in polynomial time, you prove them all in polynomial time and show that “P = NP”. Because they are all just different versions of the same problem.**

*Please feel free to re-write these answers as I am not 100% sure on them*

**Examples of NP complete problems:**

**The Hamilton Cycle problem.**

**The Bin-Packing problem.**

**0-1 Knapsack problem**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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 logn)**

n^2 - worst case is already sorted list **- yep, my bad O(n log n) is the average case.**

B) Merge Sort

**O(n log**(base2) **n)**

C) Searching an unordered list

**O(n) [unordered list would be the same as an unordered array? -** yes, the distinction here is with an ordered list/array where you can use binary search**] An ordered list is still O(n), you can’t use binary search on a list. (List refers to a linked list)**

D) Binary Search

**O(log**(base 2) **n)**

E) Searching a Perfect Hash Table

**O(1)**

F) Searching a Binary Search Tree

**O(n) - could be severely unbalanced and becomes a linked list.**

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.

**I got...  
  
f(n-1) = (n-1) \* f(n-2)  
f(n-2) = (n-2) \* f(n-3)  
…  
  
Subbing into main equation...  
f(n) = n \* (n-1) \* f(n-2)  
f(n) = n \* (n-1) \* (n-2) \* f(n-3)  
…  
  
So we get the recurrence equation (with a series within the equation) of...  
f(n) = n \* (n-1) \* (n-2) \* . . . \* (n - (k - 1)) \* f(n - k)  
  
If we set k = n -1...  
f(n) = n \* (n-1) \* (n-2) \* . . . \* (n - (n - 2)) \* f(n - (n - 1))**

**f(n) = n \* (n-1) \* (n-2) \* . . . \* (2) \* f(1)**

**f(n) = n \* (n-1) \* (n-2) \* . . . \* 2 \* 1  
  
The factorial function n! can be defined as...  
n! = n \* (n-1)!  
n! = n \* (n-1) \* (n-2)!  
n! = n \* (n-1) \* (n-2) \* . . . \* 2 \* 1  
  
So the function f(n) = n \* f(n-1) where f(1) = 1 defines the function n!.**

Yeah you are right, i recalculated it, your work seems correct to me too. :P

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

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

Don’t really know what I’m doing.

There should be constants c, n0 such that n2 < Cn! for n > n0

If we take C = 1, n = 4:

n2 = 16, n! = 24 So inequality holds for n > 4.

Assuming that for some arbitrary value, k (chosen from the set of natural numbers greater than or equal to 4), the inequality holds (k2 < k!), does it hold for k+1? (So, (k+1)2 < (k+1)!)

(k+1)2 ***=*** k2 + 2k + 1

***<=*** k2 + 2k + k (k is at least 4, inequality holds)

k2 + 3k

k2 + k2 (k2 is always bigger than 3k for k > 3)

2k2

2k! (k! > k2 for k > 3)

(k+1)k! (k is at least 4, according to our assumption, so k+1 must

be greater than k and obviously, 2. Inequality holds).

(k+1)! (By definition of a factorial).

***It looks good to me. You should try setting out like in the example at the end of the document, though. And mention that k^2 < k! is your* Induction Hypothesis.**

**2010 EXAM QUESTION 4: Sorting**

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

See 2011 exam.

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?

Quicksort. Both Merge Sort and Quicksort have O(n log n) ***average*** case performance, and since we’re working with a large data collection there’s no way we’d use insertion sort (but we could still degrade to it). Since memory is a big concern, we shouldn’t use Merge Sort as it makes new arrays when it divides up the array [technical point - I think Merge Sort makes the new array when it does the merge operation. When the array is being divided it is just done by moving the indexes to start, mid and end in the original list], therefore we should use Quicksort which is in-place (all operations take place in the one array).

***It’s for a search engine so I assumed that malicious data (ie sorted words or one that exploits median of three) could be received, maybe put on a website or something. So quicksort could be degraded to O(n^2) quite easily and you’d need a random pivot to avoid this.***

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

Set up the “buckets”:

0.00-0.09, 0.10-0.19, 0.20-0.29, 0.30-0.39, 0.40-0.49, 0.50-0.59, 0.60-0.69, 0.70-0.79, 0.80-0.89

Assign the data to the appropriate bucket:

0.00-0.09:

0.10-0.19: 0.11

0.20-0.29

0.30-0.39: 0.34

0.40-0.49

0.50-0.59: 0.54, 0.50

0.60-0.69

0.70-0.79: 0.71

0.80-0.89: 0.88, 0.81

Sort the elements within each bucket:

0.00-0.09:

0.10-0.19: 0.11

0.20-0.29

0.30-0.39: 0.34

0.40-0.49

0.50-0.59: 0.50, 0.54

0.60-0.69

0.70-0.79: 0.71

0.80-0.89: 0.81, 0.88

Combine the buckets:

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

**Willem said this won’t be in the exam (with decimals, anyway)**

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

**See 2011 exam.**

**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:

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

**I get:**

**h(34) = 6**

**h(54) = 5**

**h(88) = 4**

**h(81) = 4**

**h(11) = 4**

**h(50) = 1**

**h(71) = 1**

(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.

* **g(81) = 1 + (81 % 6) = 1 + 3 = 4**
* **g(11) = 6**
* **g(50) = 3**
* **g(71) = 6**

**We need to calculate g(k) for all the collisions, and we need to calculate it for 50 since it will collide with 81 since g(81) leads it into 50’s home cell.**

**So I think this is what you get... 71 has to cycle through the table with a step of 6 quite a few times before it can land in slot 2. Please check haha.**



I get the same answer.

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

See 2011 exam.

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?

See 2011 exam.

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.

See 2011 exam.

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

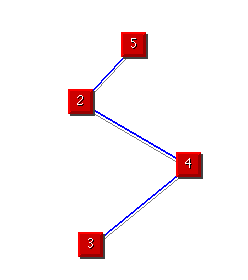
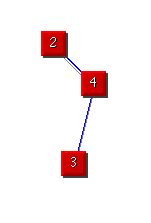
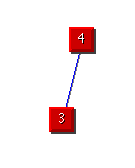
A) Draw the final binary search tree T that results from successively inserting the keys 1, 5, 2, 4, 3 into an initially empty tree.



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.

thenthen 

http://www.cs.jhu.edu/~goodrich/dsa/trees/btree.html

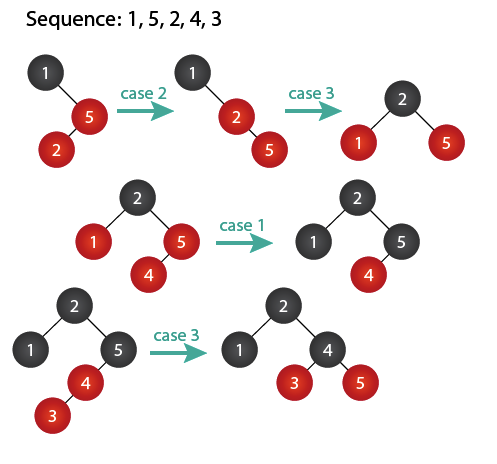
D) Give one reason why you might choose to store data in a binary search tree instead of a hash table.

See 2011 exam.

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

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

I think I get how these are constructed, they haven’t been checked so please make sure I’m right first.



**Looks good.**

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

**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.

See 2011 exam.

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

See 2011 exam.

**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).

See 2011 exam.

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).

See 2011 exam.

**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).

|  |  |  |
| --- | --- | --- |
| j | 1 | 2 |
| F1[j] | 7 | 7 |
| F2[j] | 5 | 11 |

|  |  |
| --- | --- |
| j | 2 |
| L1[j] | 2 |

F\* = 12

L\* = 2

Can someone please confirm?

***I get the same answer :)* Yup, same.**

**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.

See 2011 exam.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**2009 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) Insertion Sort

O (n^2)

B) Counting Sort

O (n)

C) Radix Sort

O (n)

D) Binary Search

**O(log** **n)**

E) Searching a Perfect Hash Table

**O(1)**

F) Searching a Red-Black Tree

**O(log** **n)**

G) Searching all subsets of a set (Isn’t it generating all subsets?)

O (2^n)

**2009 EXAM QUESTION 2: Big-O**

A) Is 3n^3 + 2n^2 + n + 1 = O(n^3)? Show your reasoning.

???

B) Is n^3 = O(3n^3 + 2n^2 + n + 1)? Show your reasoning.

???

**2009 EXAM QUESTION 3: Induction and Recurrences**

Using induction, prove that the recurrence

f(1) = 0

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

defines the function f(n) = n(n-1)/2

**Recurrence:**

**f(1) = 5**

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

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

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

**f(n) = f(n - 4) + (n - 4) + (n - 3) + (n - 2) + (n - 1)**

Huh? you aren’t supposed to do a recurrence....

**we can see that f(n) = f(n - k) + n(n - 1) / 2**

**simplify by removing k (k = (n - 1)) gives:**

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

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

**= n(n - 1)/ 2**

I did it like this:

We want to show that f(n) = g(n) for all n >= 1

Does n = 1 work?

f(1) = 0

g(1) = 1(1-1)/2 = 0

f(1) = g(1)

Assume f(k) = g(k) -> induction hypothesis

= k(k-1) / 2

Does k+1 work?

We want to show that f(k + 1) = (k+1)(k+1-1) / 2 for all n >= 1

= k(k+1)/2

f(k+1) <= f(k+1-1) + (k+1-1)

<= f(k) + k

<= k(k-1) / 2 + k f(n) = f(n-1) + (n-1)

g(n) = n(n-1) / 2

by I.H

<= (k^2/)2 - (k/2) + (2k/2) -> expanded

<= k(k+1) / 2

**I got what blue got.**

**2009 EXAM QUESTION 4: Sorting**

A) For both Merge sort and Quicksort, what are the memory requirements, what is the worst-case time complexity, and what is the average-case time complexity?

Memory requirements for Mergesort: O(n)

Average case complexity for Mergesort: O(n log n)

Worst case complexity for Mergesort: O(n log n)

Memory requirements for Quicksort: O(log n) because Quicksort is an in-place sorting algorithm

Average case complexity for Quicksort: O(n log n)

Worst case complexity for Quicksort: O(n^2)

B) How does one “randomise” Quicksort, and why would one choose to?

If good and bad pivots are chosen at random during Quicksort, the overall effect is as if they were good. Quicksort can be randomised by using a random number generator to pick a pivot and swap it with the item at A[0], without changing Partition itself. We can then expect the split of the input array to be fairly balanced most of the time.

C) Draw the bucketing structure produced by Bucket Sort when sorting the following data: 0.86, 0.96, 0.34, 0.11, 0.26, 0.73.

See 2010 exam.

D) What is the difference between a stable sort and an unstable sort? Why does Radix sort require a stable sort as its subsidiary sorting method?

See 2011 exam.

**2009 EXAM QUESTION 5: Hash Tables**

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

**I get:**

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

Are other people getting this?

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

See 2011 exam.

(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.

See 2011 exam.

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

See 2011 exam.

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?

See 2011 exam.

C) Why is it important to choose a prime number as the capacity of a hash table (particularly one that uses double hashing)?

See 2011 exam.

**2009 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.

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

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

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

See 2011 exam.

**2009 EXAM QUESTION 7: Red-black Trees**

A) Show all the red-black trees that result after successively inserting the keys 20, 15, 10, 5, 1, 3 into an initially empty red-black tree.



Mine is slighty different - 3 and 1 are reveresed and 3 is the inner child of one. When is added it is the child of 1 and its uncle (10) is red which is a case 1 and 5’s black is pushed down to 3 and 10. Could be wrong though.... **Ahh yes you’re right, I applied a case 2 to fix the zig-zag but the uncle was red, so it should have dragged the black down. This should be the right tree?**



B) Show all the red-black trees that result from the deletion of 5.



**Cbf drawing all the trees, but this should be the final one...Involved a Case 4 fix up.**

Yup - looks good to me **Cool, looks like this wasn’t affected by the mistake above. A case 3 occurs first to get it into a case 4, then it’s the same.**

**2009 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 6, 2, 4, 8, 7, 9, 10, 11, 12 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. (5)

See 2011 exam.

B) Show what happens when you delete first key 8 and then key 9.

See 2011 exam - strengthen the node via merging and borrowing prior to deletion.

**2009 EXAM QUESTION 9: 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)

The space requirements of the adjacency matrix are high - if the graph is dense this space is utilised, if the graph is sparse an adjacency list would be more economical. (Dense means the number of edges in E is close to n2, sparse means n or fewer edges.)

Also depends on the operations we want to use. Apart from insertion and deletion, the three most common operations on graphs are:

• Given two vertices j and k, determine whether there is an edge connecting them.

• Given vertex j, find all vertices adjacent to j.

• Given a vertex j as starting point, traverse the graph.

The first operation is supported best by the adjacency matrix, the second and third by the adjacency list.

B) Copy the following directed acyclic graph into your answer book. Do a depthfirst

search, showing time stamps, and then topologically sort the data.

Topological sorting involves running a depth first search and then listing the nodes in reverse order of finishing time.

C) Copy the following undirected graph into your answer book. Show how Prim’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. Draw the minimal spanning tree.

For each node, parent and cost values should be initialised to infinity and -1, respectively.

I got the order of extraction to be:

A,G,B,C,E,D

What did you all get?

6

I’m going to try draw my minimal spanning tree...

G -- F

/

A E

\ /

B -- C

\

D

**2009 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.

**2009 EXAM QUESTION 11**

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.

See 2011 exam.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**2008 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) Merge sort

O(n log n)

B) Binary search

O (log n)

C) Insertion into a hash table that uses chaining

O(1)

D) Counting sort

O(n)

E) Insertion sort

O(n^2)

F) Quicksort

O(n^2)

G) Insertion into a red-black tree

O(log n)

**2008 EXAM QUESTION 2: Recurrences**

Show by the iteration method that, if

f(1) = 0 and

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

then

f(n) = 1 +2 + …+ (n-1)

You do not need to prove your solution correct.

**2008 EXAM QUESTION 3: Big-O and Induction**

Prove that

n^3 = O(n!)

we take c, n0 such that n^3 <= c.n! for all n >= n0

(Atom) take c = 1, n0 = 6.

Does 6 work? 6^3 = 216, 6! = 720

(Closure) Assume that k works, ie. k^3 <= k!. Does k + 1 work?

(k + 1)^3 = (k + 1)(k + 1)(k + 1) [expand brackets]

= (k + 1)(k^2 + 2k + 1) [expand brackets]

= (k^3 + 3k^2 + 3k + 1) [expand brackets]

= k^3 + 3k^2 + 4k [k >= 1 so we substitute]

= k^3 + 4k^2 [k^2 >= 4k so we substitute]

= 2k^3 [k^3 >= 4k^2 so we substitute]

= 2.k! [using the induction hypothesis]

= (k + 1).k! [(k + 1) >= 2 so we substitute]

= (k + 1)! [by definition of factorial]

Inequalities...?

**2008 EXAM QUESTION 4: Sorting**

A) Compare and contrast the Merge sort and Quicksort algorithms. How are they similar? How are the different?

See 2011 exam.

B) Draw the bucketing structure produced by Bucket Sort when sorting the following data: 0.25, 0.95, 0.69, 0.80, 0.88, 0.79, 0.56.

See 2010 exam.

C) What constraint must be placed on data for Bucket Sort to have linear time complexity?

The biggest key, k, must be not much bigger than the total number of keys, i.e. k = O(n).

D) What two features are required of Radix Sort’s subsidiary sorting method? Name a sorting method that meets those requirements.

Radix Sort requires a stable sorting method with linear complexity (i.e. one that is not a comparison method). Counting sort meets these requirements.

**2008 EXAM QUESTION 5: Hash Tables**

A) Given a table of size 7, a hash function h(k), and input key25, 95, 69, 80, 88, 79 and 56 (in that order), draw the hash table that results from:

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

See 2011 exam.

(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.

See 2011 exam.

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

See 2011 exam.

B) Suppose you were using a perfect hashing scheme to create a hash table from the keys above. Would h(10,10) as defined above be acceptable as the primary hash function? Why or why not?

See 2011 exam.

C) If a hash table uses double hashing to resolve collisions, what problem can occur if its capacity is not a prime number?

See 2011 exam.

**2008 EXAM QUESTION 6: Binary Search Trees**

A) Draw the final binary search tree T that results from successively inserting the key 12, 10, 9, 7, 8, 5, 6, 3 into an initially empty tree.

See 2011 exam.

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

See 2011 exam.

C) Draw the results of deleting 9 from T.

See 2011 exam.

D) Give two reasons why, for some applications, you might prefer to use a binary search tree instead of a hash table.

See 2011 exam.

**2008 EXAM QUESTION 7: Red-black Trees**

A) Show all the red-black trees that result after successively inserting the keys 12, 10, 9, 7, 8, 5, 6, 3 into an initially empty red-black tree.

B) Show all the red-black trees that result from the successive deletion of the keys 9 and 7.

C) Describe why you might choose to use a red-black tree instead of an ordinary binary search tree.

**2008 EXAM QUESTION 8: B-trees**

B-trees of minimum degree t=2 are called 2-3-4 trees. 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.

See 2011 exam.

**2008 EXAM QUESTION 9: Graphs**

A) Copy the following directed graph into your answer book. Use depth-first search to find the strongly connected components of the diagraph. Show the finishing times computed by the first application of the depth-first search. Start at key value a and consider adjacency lists to be alphabetically ordered.

See 2011 exam.

B) Copy the following weighted undirected graph into your answer book. Show how Dijkstra’s algorithm would find the shortest paths from source a. Remember to record all priority values and to show the order in which vertices are extracted from the priority queue.

See 2011 exam.

**2008 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.

**2008 EXAM QUESTION 11: P and NP**

A) Describe the classes P and NP, and what it means to say that a problem is NP complete.

See 2011 exam.

B) Assume that some genius has proved what we all believe, namely that there are some problems in NP that do not belong to P. Would there be any NP-complete problems in P? Give reasons for your answer.

???

C) Continue to assume that P =/= NP. Let Sort be the problem of sorting an array. Would Sort belong to the class of NP-complete problems? Give reasons for your answer.

???

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

