The topics on the final exam will be drawn entirely from the following questions. The particular questions may differ, but the topics will not.

Exam 1 Questions

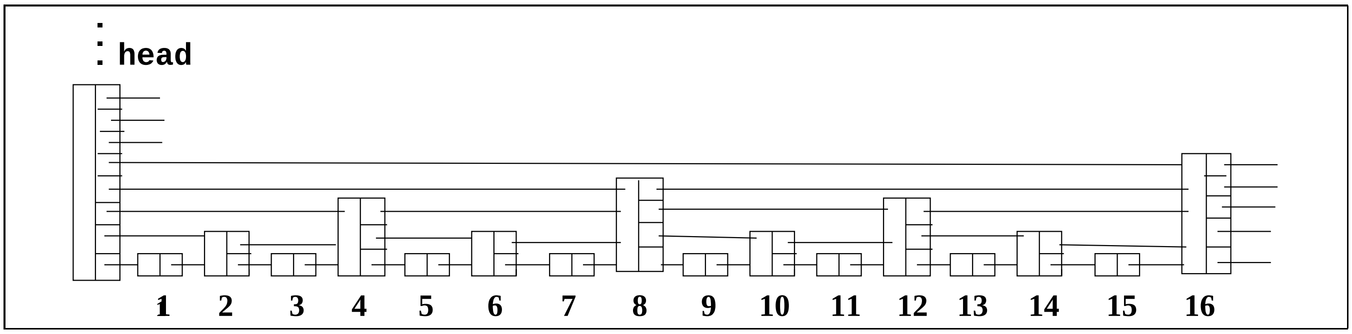
1. Every question from Exam 1. And with a somewhat lower probability, from the exam review questions.

Exam 2 Questions

2. Every question from Exam 2. And with a somewhat lower probability, from the exam review questions.

Skip List

3. The following perfect skip-list is valid for $p = \frac{1}{2}$. Draw an equivalent figure for $p = \frac{1}{4}$. What distribution of node levels do you expect in a long list of this type?



4. The expected asymptotic time performance for `SkipList` operations is $O(\lg n)$. There is a non-zero probability that the performance could be as bad as $O(n)$. Draw a 7 element `SkipList`, with `int` data values, that would have such poor performance. Use a maximum node level of 4.

5. What maximum node size is appropriate for a `SkipList` suitable for storing up to 65,536 elements and with associated probability $\frac{1}{4}$.

6. Write pseudo-code for the `find(const Comparable &)` operation in a skip-list. Return the element found if it's in the list, `ITEM_NOT_FOUND` otherwise.

7. Given a skip list (a drawing of one), indicate all the comparisons done in searching for a particular element.

8. Given a skip list with probability p and maximum node size M that contains N nodes. Show the expected distribution of node sizes (*i.e.*, the number of nodes at each size).

Red-Black Tree

Note: You will be given the bottom-up or top-down rules for insertion in Red-black trees, as appropriate. The bottom-up rules may be in the form of a flow-chart or pictorial cases. The top-down rules will be in the form of pictorial cases.

9. Define *Red-Black tree*.

10. Define *black height* of a node in a red-black tree.

11. Show the result of inserting `2,1,4,5,9,3,6,7` into an initially empty Red-Black tree (show the tree at the end of *each* insertion). Do this both bottom-up and top-down.

12. What is the "Big-Oh" performance (in terms of the number of nodes in the tree) for each operation `find`, `insert`, and `remove` for Red-Black trees in the worst and average cases?

13. What property of Red-Black trees is most significant in explaining their "Big-Oh" behavior for the operations `find`, `insert`, and `remove`.

14. Prove: in any red-black tree, no path from any node N to a leaf is more than twice as long as any other path from N to any other leaf.

15. Prove: a red-black tree with n internal nodes has height $h \leq 2\lg(n+1)$.

16. Prove: in any red-black tree, at least half of the nodes on any path from root to a leaf must be black.

17. Prove: any red-black tree with n internal nodes has height $h =\leq 2\lg(n+1)$.

18. Prove: any red black with root x has at least $n = 2^{\mathrm{bh}(x)} - 1$ internal nodes, where $\mathrm{bh}(x)$ is the black height of node x.

B-Tree

19. Define B-Tree.

20. Give pseudo-code for search in a B-Tree of order M.

21. Given a drawing of a B-Tree, show the tree after insertion of a given element.

22. Given a drawing of a B-Tree, show the tree after deletion of a given element.

23. Draw a B-tree with M = 4 and L = 3 containing the integer values 1 to 25.

24. Show the result of inserting the elements `1,3,7,9,5,11,13,6` into an initially empty B-tree having $M = 3$ and $L = 3$. Show the tree at the end of *each* insertion. Assume the key of an element is equal to the element. By definition, M is the order of the B-tree and L is the maximum number of elements that can be stored in a leaf node.

25. Given some characteristics of an external storage problem:
    (a) The number of items to be stored.
    (b) The size (in bytes) of the key for each item.
    (c) The size (in bytes) of each item.
    (d) The block size (in bytes) of a disk block.

    design a suitable B-tree (give its order, M and leaf size, L).

Priority Queues and Heaps

Note: You will be given the algorithm for merging leftist heaps.

26. Define “priority queue.”

27. Define “binary heap.”

28. Define “null path length” in a binary tree.

29. Define “leftist binary tree.”

30. Define “leftist heap.”

31. Insertion and deletion in a binary heap is in $O(lg\, n)$ on average. Explain why this is so.

32. Finding the minimum element in a binary heap (a min-heap) is in $O(1)$ worst case. Explain why this is so.

33. Prove that the largest element in a min binary heap is a leaf.

34. For a min binary heap of `N` elements, what is the range of indices in which the maximum element will be found?

35. Describe, in English, an algorithm to find the largest element in a binary min-heap. What is the asymptotic worst case performance of your algorithm?

36. The array representing a binary heap contains the following elements in the order $2, 8, 3, 10, 16, 7, 18, 13, 15$. Show the order that results at **each** stage of inserting the element 4.

37. The array representing a binary heap contains the following elements in the order $2, 8, 3, 10, 16, 7, 18, 13, 15$. Show the order that results at **each** stage of deleting the minimum element.

38. Describe, in English, the process for constructing a binary heap from a given set of initial values (*i.e.*, heapify an array of elements).

39. Construct a binary heap using the initial values $18, 2, 13, 10, 15, 3, 7, 16, 8$. Show the heap at **each** stage of construction.

40. Given a drawing of a binary tree, state if it is a leftist tree and if it is a leftist heap. Give reasons.

41. Prove that any complete binary tree is a leftist tree.

42. Prove: for any leftist tree having N vertices, the number of vertices, R, on the rightmost path to a non-full vertex is given by
$$R \leq \lfloor \lg(N+1) \rfloor$$

43. Describe how to do `findMin` in a leftist heap.

44. Using the `merge` (*aka* `meld`) operation, describe how to do the `insert` and `deleteMin` operations for leftist heaps.

45. Give pseudocode for constructing a leftist heap from a given set of values. Your code must construct a leftist heap of N elements in $O(N)$ time (**not** in $O(N \lg N)$).

46. Given drawings of two leftist heaps H1 and H2, draw the leftist heap that results from the operation `merge(H1,H2)`.

AVL Trees

Graphs

47. Define *graph*, *undirected graph*, *directed graph*, and *weighted graph*, *sparse graph.*

48. Define *path* in a graph. Define *length of a path* in a graph.

49. Define the following:

    (a) Connected, undirected graph.

    (b) Strongly connected directed graph.

    (c) Weakly connected directed graph.

50. Let $G = (V, E)$ be an undirected graph with V the set of vertices and E the set of edges. Let $v_1, v_2, \ldots, v_p \in V$ be the members of V and let $q = |E|$ be the cardinality of E. Prove:

$$\sum_{i=1}^{p} \text{degree}(v_i) = 2q$$

51. Prove that in any undirected graph, the number of vertices of odd degree is even.

52. Write pseudo-code for breadth-first and depth-first traversals of undirected graphs. The code must be complete and must fully describe the operations.

53. Describe, in English, any *adjacency table* graph implementation. How does the implementation differ for directed and undirected graphs?

54. Describe, in English, any *adjacency list* graph implementation. How does the implementation differ for directed and undirected graphs?

55. Given a drawing of a directed or of an undirected graph, show its representation as an adjacency matrix.

56. Draw the directed graph represented by the adjacency matrix given below. The rows/columns in the matrix correspond to the vertices with labels A,B,C,D,E.

```
0 1 0 1 0
1 1 1 0 0
0 0 0 0 1
0 1 0 0 1
0 0 0 0 0
```

57. Given a drawing of a directed or of an undirected graph, show its representation as an adjacency list.

58. Draw the directed graph represented by the adjacency list given below.

```
v[1] (Label = A) --> 2 --> 5

v[2] (Label = B) --> 3 --> 5

v[3] (Label = C) --> 2 --> 4 --> 5

v[4] (Label = D) --> 5

v[5] (Label = E) --> empty
```

59. Discuss the characteristics of the *adjacency table* and *adjacency list* implementations of graphs. Include storage requirements and asymptotic worst-case performance of the operations:

```
Note:  u and v are vertices in the graph

  Degree(u)       returns the degree of vertex u  (undirected graphs)
  InDegree(u)     returns the indegree of vertex u (directed graphs)
  OutDegree(u)    returns the outdegree of vertex u (directed graphs)
  AdjacentTo(u)   returns a list of the vertices adjacent to u
  AdjacentFrom(u) returns a list of the vertices adjacent from u
  Connected(u,v)  returns true if there is an edge between
                      vertices u and v, returns false otherwise
```

60. Consider the directed graph represented by the following adjacency matrix:

```
  |  A B C D E
--+------------
A |  0 1 1 1 1
B |  0 1 1 0 0
C |  0 0 0 1 0
D |  0 0 0 1 1
E |  0 1 0 0 0
```

List the depth-first and breadth-first traversals of the graph beginning at vertex A. Repeat for vertex B. (Whenever a new vertex is to be visited and there is more than one possibility, use the vertex labelled with the letter than comes first in the alphabet.)

61. Define *directed acyclic graph.*

62. Define *topological ordering* of a directed acyclic graph.

63. Given a drawing of a graph, find all cycles.

64. Given a drawing of a directed acyclic graph, write the labels of its vertices in topological order. Explain how you obtained the ordering.

65. Given a drawing of a directed graph along with the "discovery" and "finish" times of its vertices after a depth-first search, write the labels of the graph in topological order. Explain how you obtained the ordering.

66. Given a drawing of a directed graph along with the "discovery" and "finish" times of its vertices after a depth-first search, identify the type of each edge (tree, back, forward, or cross). The following test is relevant, with `d[v]` the discovery time of vertex `v` and `f[v]` its finish time:

```
if (d[v1] = d[v2] - 1)
   (v1,v2) is a tree edge
else if (d[v1] > d[v2]  && f[v1] < f[v2])
   (v1,v2) is a back edge
else if (d[v1] > d[v2] && f[v1] > f[v2])
   (v1,v2) is a cross edge
else //  d[v1] < d[v2] - 1  && f[v1] > f[v2]
   (v1,v2) is a forward edge
```

Disjoint Set

67. Give the value of the expression $lg^*(1024)$, where $lg^*(N)$ is the iterated base-2 logarithm of N.

68. Write pseudo-code to find the connected components of an undirected graph $G = (V, E)$ using Union-Find operations on the vertices.

69. Define *Union-by-Weight heuristic.*

70. Define *Path-compression heuristic.*

71. When both the union-by-weight and the path-compression heuristics are applied on disjoint sets totaling N items, a sequence of M union-find operations can be done in $O(M \lg^*(N))$ time. It is sometimes said that under these conditions, union-find is done in constant time per operation. What does this mean? Why is it true?

72. In an uptree with root x, let $R(x)$ be the length of the longest path and let $N = W(x)$ be the number of vertices (including x). Assume the uptree was created by means of multiple applications of the "union-by-size" algorithm.
Prove: $R(x) \leq \lg(N)$