University of Waterloo
CS240, Spring 2014
Assignment 5 - Update 1

Due Date: Wednesday, July 30, at 9:15am

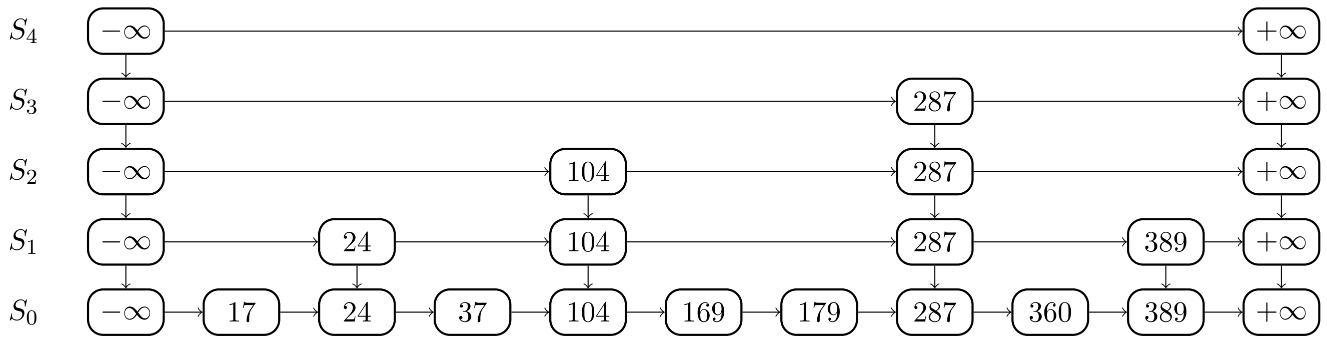
Update 1: We have added Problems 4 to 10 to cover range queries (module 7), text algorithms (module 8) and compression (module 9). All questions are written problems except problem 4.a) which is is a programming problem; submit your solution to 4.a) electronically as a file named `kdpartition.cpp`.

Please read `http://www.student.cs.uwaterloo.ca/~cs240/s14/guidelines.pdf` for guidelines on submission. Submit your solutions to written problems electronically as a PDF file with name `a05wp.pdf` using MarkUs. We will also accept individual question files named `a05q1w.pdf`, `a05q2w.pdf`, ...

Problem 1 Hashing [3+3=6 marks]

Consider a hash table dictionary with universe $U = \{0, 1, 2, \ldots, 24\}$ and size $M = 5$. If items with keys $k = 21, 3, 16, 1$ are inserted in that order, draw the resulting hash table if we resolve collisions using:

Problem 2 Skip Lists [6+6+8=20 marks]



a) Consider the skip-list S shown above. Show how $Search(S, 360)$ and $Search(S, 17)$ proceeds. More specifically show at which order search visits the nodes of the skip list. Give only the successful search path, that is only the nodes that result into a 'go right' or 'go down'. You should refer to the nodes using their keys and levels, e.g., you can say node 104 at level 1. The lowest level is 0.
Search(S, 360)

(a) node -∞ at level S_4

(b) node -∞ at level S_3

(c) node 287 at level S_3

(d) node 287 at level S_2

(e) node 287 at level S_1

(f) node 287 at level S_0

(g) node 360 at level S_0

Search(S, 360)

(a) node -∞ at level S_4

(b) node -∞ at level S_3

(c) node -∞ at level S_2

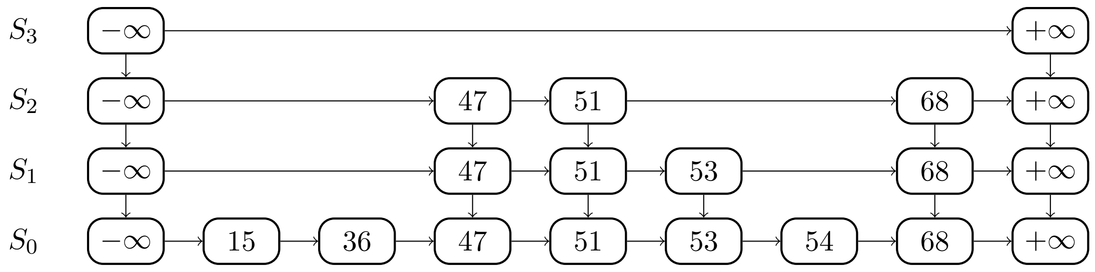
(d) node -∞ at level S_1

(e) node -∞ at level S_0

(f) node 17 at level S_0

b) Starting with an empty skip list, insert the seven keys $54, 15, 51, 53, 47, 68, 36$. Draw your final answer as on the figure in 4(a). Use the following coin tosses to determine the heights of towers (note, not every toss is necessarily used):

$$T, T, H, H, T, H, T, H, H, T, H, H, T, T, H, T, H, H, T, T, H, H, H, T, \ldots$$



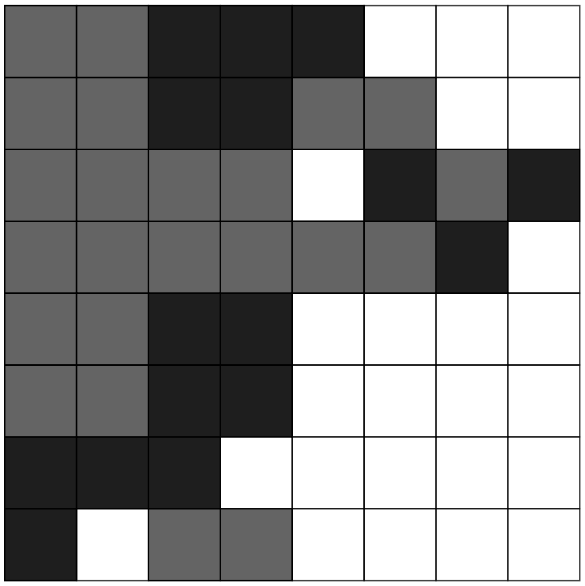
c) The worst case time for searching in a singly linked list is $\Theta(n)$. Now consider a variation of a skip list which has fixed height $h = 3$ even though n can become arbitrarily large. Level S_0 contains the keys $-\infty, k_1, k_2, \ldots, k_n, \infty$. Level S_3 contains only $-\infty$ and ∞. Describe subsets of keys that should be included in levels S_1 and S_2 so that searching in the skip list has worst case cost $\Theta(n^{1/3})$.

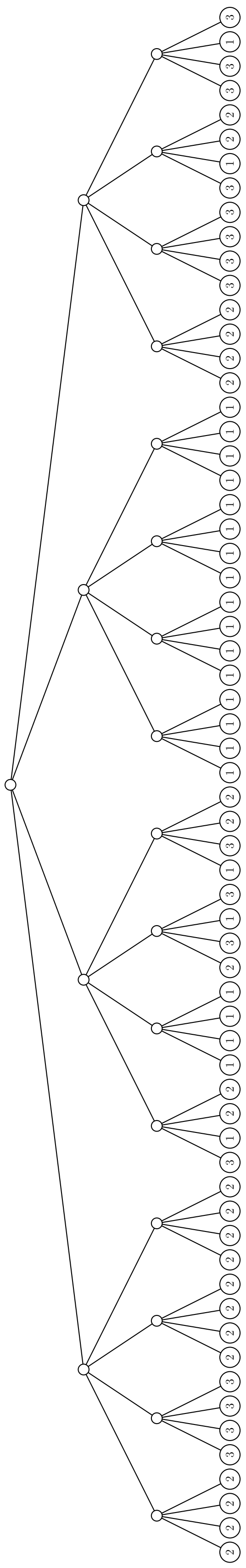
The S_2 layer contains multiples of $n^{\frac{2}{3}}$ and the s_3 layer contains multiples of $n^{\frac{1}{3}}$ this way each level will contain at most $n^{\frac{1}{3}}$ nodes so each search will go through at most $3n^{\frac{1}{3}}$ nodes before it gets to the desired spot. This has a worst case time complexity of $\Theta(n^{\frac{1}{3}})$

Problem 3 Quad Trees [5+5=10 marks]

For both parts of this question, use the convention that each internal node of a quad tree has exactly four children, corresponding to regions NW, NE, SW and SE, in that order.

a) One of the applications of quad trees is for image compression. An image (picture) is recursively divided into quadrants until the entire quadrant is only one color. Using this rule, draw the quad tree of the following image. There are only three colors (shades of gray). For the leaves of the quad tree, use 1 to denote the lightest shade, 2 for the middle shade and 3 for the darkest shade of gray.




3
1
3
3
2
2
1
3
3
3
3
3
2
2
2
2
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
2
2
3
1
3
1
3
2
1
1
1
1
2
2
1
3
2
2
2
2
2
2
2
2
3
3
3
3
2
2
2
2

b) Give three 2-dimensional points such that the corresponding quad tree has height exactly 9. Give the (x,y) coordinates of the three points and show the quad tree. (Do not give the plane partition.)

(1,1), (2,2), (0,1024).

Problem 4 kd-Tree Construction [15+5=20 marks]

a) Implement an $O(n \log n)$ algorithm to construct a kd-tree for dimension 2. Your algorithm should read $2n + 1$ integers from standard input, separated by white space or carriage return. The first integer is the number of points. The remaining $2n$ integers are the n points themselves, according to their x and y coordinates.

Use the recipe on Slide 13 of Module 7 with the following modification on the split: If the array $Px[0..n-1]$ for $n > 1$ is storing points sorted in increasing order according to their x-coordinate, then the vertical splitting line goes through point $Px[mid]$, where $mid = \lfloor n/2 \rfloor$. The root node contains $Px[mid]$, the region to the left of the splitting line include the points $Px[0..mid-1]$, and the region the the right of the splitting line includes the points $Px[mid+1..n-1]$.

As explained in class, the idea of the algorithm is to first do some preprocessing: sort the points on both their x and y coordinates. For this preprocessing step you may use a standard library function; assume that the sorting algorithm runs in time $O(n \log n)$. You may also need some additional preprocessing. Then call a recursive function (which you create) to produce a tree like the one of Slide 14 of Module 7. Actually, your program does not need to construct the tree, but rather should just print to standard output the n points stored in the nodes of the tree in the order they are visited during an in-order traversal. The coordinates of the point must be separated by a white space and we want one point per line.

Here is an example input:

```
3
3 4
2 2
1 1
```

The points are $p_0, p_1, p_2 = (3,4), (2,2), (1,1)$. These three points correspond to the following kd-tree: