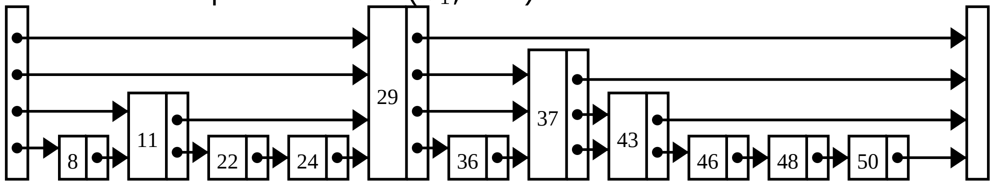
CPSC 320 SampleMidterm 2
November 2006

[12] 1. Skip Lists

[3] a. Given the skip list S_1 shown below, illustrate the contents of the `Pointer` array after the call `SkipListSearch(`S_1`, 50)`.



Solution :
Pointer[4] will point to the node with key 29
Pointer[3] will point to the node with key 37
Pointer[2] will point to the node with key 43
Pointer[1] will point to the node with key 48

[3] b. Two students Cho and Michael are inserting the same elements into the skip list from part (a). Cho is inserting them in increasing order, while Michael puts them in randomly. For some strange reason, every item inserted by Cho ends up in a node with the same level as the node with the same item from Michael's list (so the only difference is the order in which items were inserted).

What differences would you expect to see between the skip lists obtained by Cho and Michael, and why?

Solution : None: the structure of a skip list depends *only* on the levels of the nodes. The sequence of operations that were performed to get a list with the given nodes are irrelevant. This means that Cho and Michael will end up with exactly the same list.

[6] c. Explain how to modify algorithm `SkipListInsert` so that a sequence of n insertions at the *end* of the skip list can be performed in $O(n)$ expected time. Hint: you can not afford to call `SkipListSearch` to find where to insert the new node.

Solution : To achieve the desired running time, we need to reuse the `Pointer` array from the last insertion. We make sure that every element of the array always points to the last node at a given level. We then update the appropriate elements to point at the new last node1 after each insertion:

```
for i ⟵ 1 to lev do
    Pointer[i] ⟵ nextNode(Pointer[i], i)
```

where `lev` is the level of the new node. We still need to call *SkipListSearch* (or a variant of it) to setup the `Pointer` array the first time, but not afterwards.

[12] 2. Professor Kciwtilf (a rather excentric Computer Science instructor) got bored with teaching the `DeterministicSelect` algorithm the same way every time, and so one term he modified it by choosing the $\lceil 2x/5 \rceil^{th}$ order statistics of the array of x medians as the pivot, instead of the $\lceil x/2 \rceil^{th}$ element as usual.

[5] a. How many elements of the input array are guaranteed to be smaller than the pivot, using professor Kciwtilf's version of the algorithm? Justify your answer!

Solution: Let $x = \lfloor n/5 \rfloor$ groups of 5 elements. Professor Kciwtilf picks the $\lceil 2x/5 \rceil$ group, and so there are $\lceil 2x/5 \rceil$ groups that each contain at least 3 elements $\leq$ pivot (except for the group selected by professor Kciwtilf). Hence the number of elements of the input array guaranteed to be smaller than the pivot is at least

$$\begin{aligned}
3\left\lceil \frac{2\lfloor n/5 \rfloor}{5} \right\rceil - 1 &\geq 3\frac{2\lfloor n/5 \rfloor}{5} - 1 \\
&\geq 3\frac{2(n-4)/5}{5} - 1 \\
&= 3\frac{2n-8}{25} - 1 \\
&= \frac{6n-24}{25} - 1 \\
&\geq \frac{6n-25}{25} - 1 \\
&= \frac{6n}{25} - 2
\end{aligned}$$

[4] b. Derive a recurrence relation that describes the worst-case performance of professor Kciwtilf's algorithm.

Solution:
$$T(n) \leq \begin{cases} T(19n/25+1) + T(n/5) + \Theta(n) & \text{if } n \geq 10 \\ 1 & \text{if } n < 10 \end{cases}$$

[3] c. Will professor Kciwtilf's algorithm run in $O(n)$ time? Explain why or why not *briefly* (do not give a full proof; a two line justification should be about right).

Solution: Because $19n/25 + n/5 = 19n/25 + 5n/25 = 24n/25$, the extra term is linear, and $24/25 < 1$, the work done at each row of the recursion tree for $T(n)$ would look almost like a geometric series (with some irritating additional lower-order and increasingly ugly-looking terms at each level). It is thus very likely that the running time of professor Kciwtilf's algorithm is still $O(n)$.

[4] 3. We proved in class that algorithm `RandomizedQuickSelect` runs in $O(n)$ *expected* time. Why does this make sense, intuitively?

Solution: It makes sense because the probability of a good pivot (one that allows us to throw away at least $n/4$ elements) is 1/2. So we end up discarding 25% of the elements every other time which, if we were to draw the recursion tree, would give us a decreasing geometric series.

[6] 4. Professor Trahkcol (a colleague of professor Kciwtilf) decides to augment a skip list by associating with each pointer joining a node N_1 to a node N_2 at level i a value $\mathcal{P}(N_1, N_2)$ that depends on the elements of the skip list that are between N_1 and N_2. Suppose moreover that $\mathcal{P}(N_1, N_2)$ can be computed by just looking at the values associated with the level $i-1$ pointers in the interval from N_1 to N_2.

Briefly describe how the additional information (the $\mathcal{P}$ values) can be updated when a node of the skip list is deleted.

Solution : Roughly speaking, we can recompute the affected values at level i bottom-up, starting at level 1, by looking at all the level $i-1$ intervals contained in that level i interval. In pseudo-code, this would look about like:

```
for i ⟵ 1 to maxLevel(S) do
    N1 ⟵ Pointer[i]
    N2 ⟵ nextNode(N1, i)
    Go from N1 to N2 at level i-1 and compute $\mathcal{P}(N_1, N_2)$
```

Since there are $O(1)$ such intervals on average (per level), the total expected update time is $O(\log n)$.

[11] 5. A thief breaks into a pharmacy and finds various powders that he wants to place in his backpack. He finds a large book that contains, for each powder i the pharmacy has in stock: (1) the quantity q_i of powder i in stock (measured in grams), and (2) the price p_i of powder i per gram.

[8] a. Knowing that one gram of powder takes exactly 1cm^3 of space, design a greedy algorithm that will allow the thief to walk away with the contents of his backpack (which has volume V) being worth as much as possible.

Solution : The idea is to add powders in order of decreasing price per cm^3.

```
Sort the array P by decreasing value of $p_i$
i ⟵ length[P]-1
While (V > 0 and i $\geq$ 0)
    q ⟵ min ($q_i$, V)
    steals q grams of powder i
    V ⟵ V - q
    i ⟵ i -1
```

[3] b. Analyze the worst-case running time of your algorithm from part (a).

Solution : Suppose that P contains n elements. Then we need $n \log n$ time to sort the array; the remaining loop takes $O(n)$ time. Therefore the running time of the algorithm is in $\Theta(n \log n)$.