Instructions. All assignments are to be completed on separate paper. Use only one side of the paper. Assignments will be due at the beginning of class. To receive full credit, you must show all of your work.

1. Your friend has dropped you at some point on Speedway street. You need to walk to the nearest bus station, but you are not sure if the nearest bus station is East or West of your location, so you are not sure which direction to go.

Let $d$ be the distance to the nearest bus station. Suggest an algorithm that guarantees that the total distance you would walk is $\Theta(d)$. Of course, you cannot use other people, cellphones, maps etc.

Answer:

Let $x$ be the starting point. Assume the max distance from which you can see the station is 1, and assume the closest station is West of your starting point. Walk 2 units up, 4 units down, 8 up etc. If $d$ is the distance to the station then you might find yourself at distance $\leq 2$ East from $x$. So the max distance from $x$ East is

$$1 + 2 + 4 \ldots = 4d$$

and this number should be doubled for the return from this point to $x$, giving $8d$.

The distance from $x$ covered to the West until the last iteration $d/2$, summing (including returns and previous iterations) to $4 \cdot (d/2) = 2d$. Finally walked $d$ at the last iteration. So in total $O(d)$.

2. (a) Assume $S$ is a skiplist constructed on the set of $n$ keys $\{k_1, k_2, \ldots k_n\}$. Assume $X = \{x_1 \ldots x_m\}$ is a set of $m$ keys, given in a sorted order, so $x_i < x_{i+1}$ for every $i$. Also assume that $m$ is much smaller than $n$, but is not a constant.

Suggest an efficient algorithm that finds which of the keys in the SkipList are also in $X$.

Answer:

Assume we have found $x_i$ in the SL. Let $Q = \{b_1 \ldots b_k\}$ be the elements (one from each level) of the SL that were visited during the search for $x_i$. So $Key[Next[b_j]] > x_i$. Where $b_1$ is the top of the SL.

Next we search $x_{i+1}$. For this we check whether

$$Key[Next[b_1]],\ Key[Next[b_2]],\ Key[Next[b_3]]...$$

etc, till finding the the largest $k$ for which $Key[Next[b_k]] < x_{i+1}$. (In other words, $Key[Next[b_{k+1}]] > x_{i+1}$). At this point, we start a new search procedure, starting from $b_k$. 
(b) Now assume that the SkipList is perfect, in the following sense. Assume that for every level \( t \), we create this level by taking each second element from level \( t-1 \), and skipping the reminding one. So for example if level \( t-1 \) consists of keys \( \{-\infty, 1, 5, 7, 19, 24, 29, 50, 93, \infty\} \) then the keys in level \( t \) are \( \{-\infty, 1, 7, 24, 50, \infty\} \). Note that \(-\infty\) and \(\infty\) always stay. Show that for this structure, your algorithm runs in time \( O(m \log(n/m)) \).

**Answer:**

Let \( Y_i \) for \((i = 2, 3 \ldots m)\) be the highest level visited at the search for \( x_i \). Note that if \( Y_i > r \) then \( x_{i+1} - x_i \geq 2^r \). Hence

- the number of times \( Y_i > \log_2(n/m) \) (that is, the search reaches level \( \log_2(n/m) \)) cannot exceed \( 2\log_2(n/m) = n/m \).
- The number of times it reaches above \( Y_i > 1 + \log_2(n/m) \) cannot exceed \( \frac{1}{2}n/m \).
- The number of times it reaches above \( Y_i > 1 + \log_2(n/m) \) cannot exceed \( \frac{1}{2}n/m \).
- The number of times it reaches above \( Y_i > 2 + \log_2(n/m) \) cannot exceed \( \frac{1}{2^2}n/m \).
- The number of times it reaches above \( Y_i > 2 + \log_2(n/m) \) cannot exceed \( \frac{1}{2^3}n/m \).
- etc

Please use a figure to demonstrate your idea. Also provide a pseudo code of the algorithm.

3. Explain the pros and cons of SkipList vs. balanced binary search trees. **Answer:**

Pro: Very simple code. Easy to implement. No moving of data between nodes.

Extremely simple deletion

Cons: The algorithm is randomized, so there is a chance it will perform poorly for some inputs. An adversary could delete small number of elements, hampering its performances.

4. Assume \( A_1 \ldots A_n \) are events, each given with the probability \( Pr(A_i) \) that it occurs. Prove that the probability that at least one of these events happens is not larger than \( \sum_{i=1}^{n} Pr(A_i) \). Alternatively, that the probability that none of these events happens is \( \geq 1 - \sum Pr(A_i) \).

5. Consider the algorithm discussed in class for construction of SkipList, constructed for the set of keys \( \{x_1 \ldots x_n\} \). Define a random variable \( \ell_i \) to be the number of levels at which \( x_i \) participates. So for example, if \( x_\tau \) is contained in the three lower-most levels, then \( \ell_\tau = 3 \). What is \( E[\sum_{i=1}^{n} \ell_i] \)? Prove.

**Answer:**

\[
E \left[ \sum_{i=1}^{n} \ell_i \right] = \sum_{i=1}^{n} E[\ell_i] = \sum_{i=1}^{n} \sum_{j=1}^{\infty} \frac{1}{2^j} = \sum_{i=1}^{n} O(1) = O(n)
\]

6. Let \( S = \{\ell_1 \ldots \ell_n\} \) be a given set of lines, none is vertical. The upper envelop of \( S \), denoted \( U(S) \) is define as all the points of \( S \) that lie on at least one line of \( S \) and not below any of the other lines of \( S \). See Figure 1. A vertex is defined as the intersection point of two lines. The slope of a line is defined as the angle between the line and the x-axis. Assume that \( S \) is given to you such in an increasing order of slopes, so \( \ell_1 \) has the smallest slope and \( \ell_n \) as the largest slope.
Figure 1: A set $S = \{\ell_1 \ldots \ell_5\}$. Underline is their upper envelop $\mathcal{U}(S)$. The vertices on the upper envelop are marked by tiny disks.
What is the complexity of the upper envelope? (That is, how many vertices and edges it contains. Assume that to report a vertex, you need to specify two coordinates, each requires a single word. You can use big-$O$ notation).

Suggest an algorithm that computes the vertices of the boundary of the upper envelope in time $O(n)$.

**Hint** — Define $S_i = \{s_1 \ldots s_i\}$. Assume by induction that you have already computed the vertices of $U(S_i)$. How can you efficiently compute $U(s_{i+1})$?

**Answer:**

Every line contributes zero segment or one segment or half lines space to the upper envelope. Hence the complexity of upper envelope is $O(n)$. To compute the vertices of the boundary of the upper envelope in time $O(n)$ we use the induction method. We assume that we have already computed the vertices of $U(S_i)$. We copy the elements of $U(S_i)$ in to $U(S_{i+1})$. We now pick the next line $s_{i+1}$ and start scanning the set $U(S_i)$ from right to left direction (decreasing order of slopes) looking for the intersection of existing segments with this new line. We have two cases:

**Case a)** If the segment lies entirely below the line we remove it from $U(S_{i+1})$ and move to next element of $U(S_{i+1})$ in the left. Once a segment is eliminated at any stage we dont include that segment in subsequent stages of our computations.

**Case b)** If a portion of the segment lies above and below the line we include the portion of the segment above the line in $U(S_{i+1})$ and also include a new segment from the line $s_{i+1}$ which is above the segment being considered currently in $U(S_{i+1})$ and stop at this point. To prove the running time of algorithm we assume that each new line $s_{i+1}$ being considered for stage $U(S_{i+1})$ comes with 4 dollars in to the system. It uses a dollar for its insertion in to the system (i.e. it donates a dollar to the segment in case b) above). After its insertion in to the set $U(S_i)$. Every segment that is compared with the new line being tested spends its second dollar for the comparison in either cases above. If the segment is removed from $U(S_{i+1})$ as described above in case b) it utilizes another two dollars for its removal from the system. At the end of all the stages i.e. the computation of the upper envelope we would have used $4n$ dollars where $n$ is the number of lines in the set $S$. This proves that the complexity of the algorithm is $O(n)$.