Help sheet

The following facts and conventions are recalled here to help you. **Unless asked or suggested not to do so**, you have to follow the same conventions, but **you do not have to use all of the facts**.

Conventions on trees

- A single node is a tree of height 0
- The height of a node is \(1 + \max(\text{height}(L), \text{height}(R))\), where \(L\) and \(R\) are its left and right children
- In a BST, if a node stores the key \(k\) then all keys stored in its left child are \(\leq k\) and all keys stored in its right child are \(> k\)

Amortized cost

- Given a sequence of \(N\) operations, its amortized cost per operation is the overall runtime of the whole sequence divided by \(N\).
- When proving that a data structure reaches amortized cost \(c\), you must prove that any sequence of operations have amortized cost no greater than \(c\).

Mathematical facts

- We denote by \(\log\) is the logarithm in base 2, and by \(\ln\) the logarithm in base \(e\)
- We denote by \(e\) the unique real number such that \(\ln(e) = 1\), an approximation is \(e \simeq 2.7\)
- \(\sum_{i=1}^{n} i = \Theta(n^2)\)
- \(\sum_{i=1}^{n} 1/i = \Theta(\log n)\)
- If \(X\) is a random variable taking non-negative integer values then
  \[
  \mathbb{E}(X) = \sum_{i=0}^{\infty} i \mathbb{P}(X = i) = \sum_{i=0}^{\infty} \mathbb{P}(X > i)
  \]
- If \(r\) is a positive real number different from 1, then
  \[
  \sum_{i=0}^{n} r^i = \frac{1 - r^{n+1}}{1 - r}
  \]
### Some useful slides from cs240

#### Order Notation Summary

- **O-notation:** \( f(n) \in O(g(n)) \) if there exist constants \( c > 0 \) and \( n_0 > 0 \) such that \( |f(n)| \leq c|g(n)| \) for all \( n \geq n_0 \).

- **\( \Omega \)-notation:** \( f(n) \in \Omega(g(n)) \) if there exist constants \( c > 0 \) and \( n_0 > 0 \) such that \( |f(n)| \geq c|g(n)| \) for all \( n \geq n_0 \).

- **\( \Theta \)-notation:** \( f(n) \in \Theta(g(n)) \) if there exist constants \( c_1, c_2 > 0 \) and \( n_0 > 0 \) such that \( c_1|g(n)| \leq |f(n)| \leq c_2|g(n)| \) for all \( n \geq n_0 \).

- **\( o \)-notation:** \( f(n) \in o(g(n)) \) if for all constants \( c > 0 \), there exists a constant \( n_0 > 0 \) such that \( |f(n)| < c|g(n)| \) for all \( n \geq n_0 \).

- **\( \omega \)-notation:** \( f(n) \in \omega(g(n)) \) if for all constants \( c > 0 \), there exists a constant \( n_0 > 0 \) such that \( |f(n)| > c|g(n)| \) for all \( n \geq n_0 \).

#### Techniques for Order Notation

Suppose that \( f(n) > 0 \) and \( g(n) > 0 \) for all \( n \geq n_0 \). Suppose that

\[
L = \lim_{n \to \infty} \frac{f(n)}{g(n)}
\]

(in particular, the limit exists).

Then

\[
\begin{align*}
\frac{f(n)}{g(n)} &\in o(g(n)) \quad \text{if } L = 0 \\
\frac{f(n)}{g(n)} &\in \Theta(g(n)) \quad \text{if } 0 < L < \infty \\
\frac{f(n)}{g(n)} &\in \omega(g(n)) \quad \text{if } L = \infty.
\end{align*}
\]

The required limit can often be computed using l'Hôpital's rule. Note that this result gives sufficient (but not necessary) conditions for the stated conclusions to hold.

#### Useful Math Facts

**Logarithms:**

- \( c = \log_a \alpha \) means \( b^c = \alpha \). E.g. \( n = 2^{\log_2 n} \). Requires \( b > 1 \).
- \( \log(a) \) (in this course) means \( \log_a \alpha \)
- \( \log(a \cdot c) = \log(a) + \log(c) \), \( \log(a^x) = c \log(a) \)
- \( \log_a \alpha = \frac{\log \alpha}{\log \alpha_a} = \frac{1}{\log_a \alpha} \cdot \log_b \alpha = c \log_b \alpha \)
- \( \ln(x) = \text{natural log} = \log_e(x) \), \( \text{for } x = \frac{1}{a} \)
- concavity: \( \alpha \log x + (1 - \alpha) \log y \leq \log(\alpha x + (1 - \alpha) y) \) for \( 0 \leq \alpha \leq 1 \)

**Factorial:**

- \( n! := n(n - 1) \cdots 2 \cdot 1 = \# \text{ ways to permute } n \text{ elements} \)
- \( \log(n!) = \log n + \log(n - 1) + \cdots + \log 2 + \log 1 \in \Theta(n \log n) \)

**Probability and moments:**

- \( E[aX] = aE[X] \), \( E[X + Y] = E[X] + E[Y] \) (linearity of expectation)

#### Some Recurrence Relations

<table>
<thead>
<tr>
<th>Recursion</th>
<th>resolves to</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(n) = T(n/2) + \Theta(1) )</td>
<td>Binary search</td>
<td></td>
</tr>
<tr>
<td>( T(n) = 2T(n/2) + \Theta(n) )</td>
<td>Mergesort</td>
<td></td>
</tr>
<tr>
<td>( T(n) = 2T(n/2) + \Theta(\log n) )</td>
<td>Heapify ( \rightarrow \text{later} )</td>
<td></td>
</tr>
<tr>
<td>( T(n) = T(cn) + \Theta(n) ) for some ( 0 &lt; c &lt; 1 )</td>
<td>Selection ( \rightarrow \text{later} )</td>
<td></td>
</tr>
<tr>
<td>( T(n) = 2T(n/4) + \Theta(1) )</td>
<td>Range Search ( \rightarrow \text{later} )</td>
<td></td>
</tr>
<tr>
<td>( T(n) = T(\sqrt{n}) + \Theta(1) )</td>
<td>Interpolation Search ( \rightarrow \text{later} )</td>
<td></td>
</tr>
</tbody>
</table>

* Once you know the result, it is (usually) easy to prove by induction.
* Many more recursions, and some methods to find the result, in cs341.
HeapSort

* Idea: PQ-Sort with heaps.
* But: Use same input-array A for storing heap.

\[
\text{HeapSort}(A, n)
1. \quad \text{// heapify}
2. \quad n \leftarrow A.\text{size()}
3. \quad \text{for } i \leftarrow \text{parent}(\text{last}(n)) \text{ downto 0 do}
4. \quad \text{fix-down}(A, n, i)
5. \quad \text{// repeatedly find maximum}
6. \quad \text{while } n > 1
7. \quad \text{// do deleteMax}
8. \quad \text{swap items at } A[\text{root()}] \text{ and } A[\text{last}(n)]
9. \quad \text{decrease } n
10. \quad \text{fix-down}(A, n, \text{root()})
\]

The for-loop takes \(\Theta(n)\) time and the while-loop takes \(O(n \log n)\) time.

Efficient In-Place partition (Hoare)

Idea: Keep swapping the outer-most wrongly-positioned pairs.

\[
\text{partition}(A, p)
A: \text{array of size } n, \quad p: \text{integer s.t. } 0 \leq p < n
1. \quad \text{swap}(A[p-1], A[p])
2. \quad i \leftarrow -1, \quad j \leftarrow n - 1, \quad v \leftarrow A[n - 1]
3. \quad \text{loop}
4. \quad \text{do } i \leftarrow i + 1 \text{ while } i < n \text{ and } A[i] < v
5. \quad \text{do } j \leftarrow j - 1 \text{ while } j > 0 \text{ and } A[j] > v
6. \quad \text{if } i \geq j \text{ then break (goto 9)}
7. \quad \text{else swap}(A[i], A[j])
8. \quad \text{end loop}
9. \quad \text{swap}(A[n - 1], A[j])
10. \quad \text{return } i
\]

Running time: \(\Theta(n)\).

Binary Search

Ordered array
* insert, delete: \(\Theta(n)\)
* search: \(\Theta(\log n)\)

\[
\text{Binary-search}(A, n, k)
A: \text{array of size } n, \quad k: \text{key}
1. \quad \ell \leftarrow 0
2. \quad r \leftarrow n - 1
3. \quad \text{while } (\ell < r)
4. \quad \text{m} \leftarrow \lfloor \frac{\ell + r}{2} \rfloor
5. \quad \text{if } (A[m] < k) \quad \ell \leftarrow m + 1
6. \quad \text{elsif } (k < A[m]) \quad r \leftarrow m - 1
7. \quad \text{else return } \text{m}
8. \quad \text{if } (k = A[\ell]) \text{ return } \ell
9. \quad \text{else return "not found, but would be between } \ell - 1 \text{ and } \ell"
\]

Interpolation Search

* Code very similar to binary search, but compare at interpolated index
* Need a few extra tests to avoid crash due to \(A[\ell] = A[r]\)

\[
\text{Interpolation-search}(A, n, k)
A: \text{array of size } n, \quad k: \text{key}
1. \quad \ell \leftarrow 0
2. \quad r \leftarrow n - 1
3. \quad \text{while } (\ell < r) \&\& (A[\ell] = A[r]) \&\& (k \geq A[\ell]) \&\& (k \leq A[r])
4. \quad m \leftarrow \ell + \lfloor \frac{k - A[\ell]}{A[r] - A[\ell]} \cdot (r - \ell) \rfloor
5. \quad \text{if } (A[m] < k) \quad \ell \leftarrow m + 1
6. \quad \text{elsif } (A[m] = k) \quad \text{return } m
7. \quad \text{else } \quad r \leftarrow m - 1
8. \quad \text{if } (k = A[\ell]) \quad \text{return } \ell
9. \quad \text{else return "not found, but would be between } \ell - 1 \text{ and } \ell"
\]