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 \( f(n) \leq cg(n) \) 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 \( f(n) \geq cg(n) \) 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 \( c_1g(n) \leq f(n) \leq c_2g(n) \) 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 \( f(n) \leq cg(n) \) 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 \( f(n) \geq cg(n) \) for all \( n \geq n_0 \).

Useful Math Facts

Logarithms:
\( \log_b(a) \) means \( b^c = a \). E.g. \( n = 2\log_n \).

\( \log_2(n) \) (in this course) means \( \log_b(n) \).

\( \log(x + y) = \log(x) + \log(y) \).

\( \log(x^c) = c\log(x) \).

\( \log_{10}(n) = \frac{\log_e(n)}{\log_e(10)} \).

\( \log_b(n) \) is natural log \( \log_{e}(n) \).

\( \log(n) = \log_{10}(n) \).

Probability and moments:
\( E[X] = \mu[X], \ E[X + Y] = \mu[X] + \mu[Y] \) (linearity of expectation).

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
\[ f(n) \in \Theta(g(n)) \quad \text{if} \quad L \neq 0 \]
\[ f(n) \in O(g(n)) \quad \text{if} \quad L < \infty \]
\[ f(n) \in \Omega(g(n)) \quad \text{if} \quad L > \infty \]

The required limit can often be computed using \( \varepsilon \)-\( \delta \) notation. Note that this result gives sufficient (but not necessary) conditions for the stated conclusions to hold.

Some Recurrence Relations

<table>
<thead>
<tr>
<th>Relation</th>
<th>Function to</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(n) = f(n/2) + \Theta(1) )</td>
<td>Binary search</td>
</tr>
<tr>
<td>( T(n) = 2T(n/2) + \Theta(n) )</td>
<td>Merge sort</td>
</tr>
<tr>
<td>( T(n) = f(n - c) + \Theta(n) ) for some ( 0 &lt; c &lt; n )</td>
<td>Selection</td>
</tr>
<tr>
<td>( T(n) = f(n - c) + \Theta(1) )</td>
<td>Range search</td>
</tr>
<tr>
<td>( T(n) = f(n) + \Theta(1) )</td>
<td>Interpolation search</td>
</tr>
</tbody>
</table>

Once you know the result, it is (usually) easy to prove by induction.

Many more recurrences, and some methods to find the result, in cs341.
Efficient In-Place partition (Hoare)

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

\[\text{partition}(A, \ell, r)\]

- \(A\) array of size \(n\), \(\ell, r\) integer s.t. \(0 \leq \ell < n\)
- \(m = \lfloor (\ell + r) / 2 \rfloor\)
- \(\ell < m < r\)
- \(\text{while} \ (i < r)\)
  - \(m = (m + i) / 2\)
  - \(\text{if} \ (A[m] < k)\)
    - \(\ell = m + 1\)
    - \(\text{else}\)
      - \(\ell = m\)
      - \(\text{else return } \ell\)
      - \(\text{if} \ (k = A[\ell]) \text{ return } \ell\)
      - \(\text{else return "not found, but would be between } \ell - 1 \text{ and } r"\)

Running time: \(O(n)\)

Binary Search

Ordered array
- insert, delete: \(O(n)\)
- search: \(O(\log n)\)

\[\text{binary-search}(A, \ell, r, k)\]

- \(A\) array of size \(n\), \(\ell, r, k\) key
- \(f = \ell\)
- \(r = n - 1\)
- \(\text{while} \ (f < r)\)
  - \(m = (f + r) / 2\)
  - \(\text{if} \ (A[m] < k)\)
    - \(f = m + 1\)
    - \(\text{else}\)
      - \(r = m\)
      - \(\text{else return } k\)
      - \(\text{if} \ (k = A[f]) \text{ return } f\)
      - \(\text{else return "not found, but would be between } f - 1 \text{ and } r"\)

HeapSort

- Idea: PQ-Sort with heaps.
- Use same input array \(A\) for storing heap.

\[\text{HeapSort}(A, n)\]

- \(A\) array of size \(n\)
- \(n = A[A[0]]\)
- \(\text{for } i = 1, \ldots, n\)
  - \(\text{do}\)
    - \(A[i] = \text{parent}(A, i)\)
  - \(\text{end for}\)
- \(\text{while} \ n > 1\)
  - \(\text{do}\)
  - \(\text{end while}\)
- \(\text{return } A[1]\)

Running time: \(O(n)\) time and the while-loop takes \(O(n \log n)\) time.

Count Sort Pseudocode

\[\text{key-indexed-count-sort}(A, d)\]

- \(A\) array of size \(n\), contains numbers with digits in \([0, \ldots, R - 1]\)
- \(d\) index of digit by which we wish to sort
- \(\text{count}\) how many of each kind there are
  - \(\text{for } i = 0 \text{ to } n - 1\)
  - \(\text{for } j = 0 \text{ to } R - 1\)
  - \(\text{if } A[i] < j\)
    - \(\text{count}[j] = \text{count}[j] + 1\)
  - \(\text{return } \text{count}\)

Interpolation Search

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

\[\text{interpolation-search}(A, A, k)\]

- \(A\) array of size \(n\), \(k\) key
- \(\text{while} \ (f < k)\)
  - \(m = \lfloor (r - f) (k - f) \rfloor / (r - f)\)
  - \(\text{if} \ (A[m] < k)\)
    - \(f = m + 1\)
    - \(\text{else}\)
      - \(r = m\)
      - \(\text{else return } k\)
      - \(\text{if} \ (k = A[m]) \text{ return } m\)
      - \(\text{else return "not found, but would be between } f - 1 \text{ and } r"\)