CSE 3500 Algorithms and Complexity – Fall 2016
Lecture 7: September 20, 2016

Binary Search

- Input: A sorted sequence $X = k_1, k_2, \ldots, k_n$ and another element x; Output: 'yes' if $x \in X$ and 'no' otherwise.
- The binary search algorithm works as follows:

BinarySearch(X, i, j, x)
  if $i = j$ **then if** $x \neq k_i$ **then** output 'no' and quit;
  if $x = k_{(i+j)/2}$ **then** output 'yes' and quit;
  else if $x < k_{(i+j)/2}$ **then** BinarySearch$(X, i, (i + j)/2 - 1, x)$;
  else BinarySearch$(X, (i + j)/2 + 1, j, x)$;

Merge sort

- Merge sort takes as input a sequence $X = k_1, k_2, \ldots, k_n$ and outputs X in sorted order. This is a divide-and-conquer algorithm with the following steps:

0.1) **if** $n = 1$ **then** output k_1 and quit;
0.2) **if** $n = 2$ **then** { **if** $k_1 < k_2$ **then** output k_1, k_2 and quit;
  else output k_2, k_1 and quit; }
1) Let $X_1 = k_1, k_2, \ldots, k_{n/2}$ and $X_2 = k_{n/2+1}, k_{n/2+2}, \ldots, k_n$;
2) Recursively sort X_1 to get Y_1; Recursively sort X_2 to get Y_2;
3) Merge Y_1 and Y_2;

- **Merging two sorted sequences:** Let $A = a_1, a_2, \ldots, a_n$ and $B = b_1, b_2, \ldots, b_m$ be two sorted sequences. The problem of merging A and B is that of producing a sorted sequence C that has all the elements of A and B.
- We can merge two sorted sequences as follows: Compare the smallest element of A with the smallest element of B, output the smaller of these two elements, and delete this element from its sequence; Repeat this step until one of the sequences becomes empty at which point output all the remaning elements from the other sequence.

- An example: Let $A = 2, 5, 8, 12, 32, 45, 50$ and $B = 3, 4, 9, 10, 20, 22, 26, 27, 41$. We compare 2 from A and 3 from B and output 2. We delete 2 from A. Followed by this, we compare 5 from A with 3 from B and output 3. Subsequently 3 is deleted from B; and so on. When 41 is output from B, two elements $45, 50$ will remain in A. These two elemens are finally output.
- Note that the number of comparisons made by the above merging algorithm is no more than $n + m - 1$. This is because whenever we make a comparison we output one element. There are at most $n + m$ elements that we have to output. Also, at some point when one of the sequences becomes empty, the other sequence will have at least one element (and we do not have to compare this element with any other element).

Run time analysis of divide-and-conquer recursive algorithms

- For a generic divide-and-conquer recursive algorithm we let $T(n)$ be the run time on any input of size n. We then account for the run time of each step in the algorithm and add up all of these run times to compute $T(n)$. Some of the steps could be recursive and hence their run times will be expressed using the funtion $T(.)$ itself. Thus we will express the function $T(.)$ in terms of itself. Any such releation is called a Recurrence Relation. We have to solve this recurrence relation to get the run time of the algorithm (in a form that we can readily appreciate).
- **The example of binary search:** Let $T(n)$ be the run time of binary search on any input of size n. Then, we can express $T(n)$ as follows: $T(n) = T(n/2) + 1$.
- **The example of merge sort:** Step 1 of merge sort does not need any comparisons. In step 2 we have two recursive calls each on an input of size $n/2$. Thus the total time spent for step 2 is $2T(n/2)$. Step 3 takes no more than n comparisons (as we have seen before). Thus we end up with the following:

$$T(n) \leq 2T(n/2) + n.$$

- There are many ways of solving a recurrence relation. We will discuss three such techniques.

Solving a recurrence relation by repeated substitutions

- Consider the following recurrence relation:

$$T(n) = \begin{cases} c & \text{if } n \leq d \\ aT(n/b) + f(n) & \text{if } n > d \end{cases}$$

where $a > 0, b > 1, c \geq 0$, and $d \geq 0$ are integer constants.

- Our goal is to eliminate the occurrence of the function $T(.)$ from the right hand side (RHS). In order to do so, we repeatedly substitute for $T(n/b)$ using the definition of $T(n)$. Whenever we make a substitution, the value of the parameter for $T(.)$ monotonically decreases. This value will eventually become $\leq d$ (i.e., the base case) at which point we would have eliminated the occrrence of the function $T(.)$ from the RHS.
- We will illustrate this technique with examples.
- **The example of merge sort:** Consider the recurrence relation:

$$T(n) = \begin{cases} 1 & \text{if } n \leq 2 \\ 2T(n/2) + n & \text{if } n > 2 \end{cases}$$

- We start with $T(n) = T(n/2) + n$ and make a substitution for $T(n/2)$:

$$T(n) = 2[2T(n/4) + n/2] + n = 2^2T(n/2^2) + 2n.$$

  Making one more substitution we get:

$$T(n) = 2^2[2T(n/2^3] + n/2^2] + 2n = 2^3T(n/2^3) + 3n.$$

- We now see a general pattern. If we make $(i-1)$ substitutions (for any $i > 1$) we will obtain:

$$T(n) = 2^iT(n/2^i) + in. \tag{1}$$

- If $(n/2^i) = 2$ (corresponding to the base case), we will be able to eliminate the occurrence of $T(.)$ from the RHS.
- $(n/2^i) = 2$ when $n = 2^{i+1}$, i.e., when

$$i = (\log n - 1). \tag{2}$$

- Substituting equation 2 in equation 1, we notice:

$$T(n) = \frac{n}{2}T(2) + (\log n - 1)n = n\log n - \frac{n}{2} = \Theta(n\log n).$$

- In summary, the solution to the recurrence relation is $T(n) = \Theta(n\log n)$.

Another example

- Consider the following recurrence relation:

$$T(n) = \begin{cases} 1 & \text{if } n \leq 2 \\ 5T(n/2) + n^2 & \text{if } n > 2 \end{cases}$$

- We start with $T(n) = 5T(n/2) + n^2$ and make a substitution for $T(n/2)$:

$$T(n) = 5[5T(n/4) + (n/2)^2] + n^2 = 5^2T(n/2^2) + (5/4)n^2 + n^2.$$

  Making one more substitution we get:

  $T(n) = 5^2[5T(n/2^3) + (n/2^2)^2] + (5/4)n^2 + n^2 = 5^3T(n/2^3) + (5/4)^2n^2 + (5/4)n^2 + n^2.$

- Making yet another substitution:

  $T(n) = 5^3[5T(n/2^4)+(n/2^3)^2]+(5/4)^2n^2+(5/4)n^2+n^2 = 5^4T(n/2^4)+(5/4)^3n^2+(5/4)^2n^2+(5/4)n^2+n^2$

- We now see a general pattern. If we make $(i-1)$ substitutions (for any $i > 1$) we will obtain:

$$T(n) = 5^iT(n/2^i)+[(5/4)^{i-1}+(5/4)^{i-2}+\cdots+(5/4)+1]n^2 = 5^iT(n/2^i)+\frac{(5/4)^i-1}{(5/4)-1}n^2. \quad (3)$$

- (We have used the fact that $1 + r + r^2 + \cdots + r^k = \frac{r^{k+1}-1}{r-1}$ for any $r \neq 1$). If $(n/2^i) = 2$ (corresponding to the base case), we will be able to eliminate the occurrence of $T(.)$ from the RHS.

- $(n/2^i) = 2$ when $n = 2^{i+1}$, i.e., when

$$i = (\log n - 1). \quad (4)$$

- Substituting equation 4 in equation 3, we notice:

$$T(n) = 5^{\log n-1}T(2) + 4[(5/4)^{\log n-1} - 1]n^2 = \frac{1}{5}n^{\log 5} + \left[\frac{16}{5}(5/4)^{\log n} - 4\right]n^2$$

$$= \frac{1}{5}n^{\log 5} + \frac{16}{5}5^{\log n} - 4n^2 = \frac{1}{5}n^{\log 5} + \frac{16}{5}n^{\log 5} - 4n^2 = \Theta(n^{\log 5}).$$

- (Here we have used the following facts: $4^{\log n} = n^2$ and more generally, $a^{\log_b n} = n^{\log_b a}$). In summary, the solution to the recurrence relation is $T(n) = \Theta(n^{\log 5})$.