CSCI 3104 Spring 2014 Problem Set 2
Dennis Johnson (06/02)
Max Harris (07/01)

---

1. We can analyze this algorithm with a sorted list of length n and recognize that it is a binary search algorithm.

   As the search for key k iterates over i in sorted list n, the size of list $n_i = n/2^i$ until either k is found and returns the element or k is not found and the list size becomes 0. In this sense, as n increases, the set of elements is decreasing at an exponential rate. We can solve for i to determine the max amount of iterations.

$$n_i = \frac{n}{2^i} => 2^i n_i = n \tag{1}$$

   Take lg of both sides

$$\lg(2^i n_i) = \lg(2^i) + \lg(n_i) = i \;\; \lg(2) + \lg(n_i) = i + \lg(n_i) = \lg(n) \tag{2}$$

   We can see that the number of iterations i that this algorithm requires is on the order of $\lg(n)$.

   (a) *Using Θ notation, what is the best case running time as function of n?*
   In the best case situation, we look at the minimum number of operations involved. The best case in this situation is when the key k we are looking for is at index $n/2$. This is the first key the algorithm evaluates and only evaluates once. The other operations are constant operations and involve assignments and comparisons. There are a finite number of them and it is not based on the size of n. For instance, we assign $x = 0$ and $y = len(a)$.

   In this sense, there are constant operations that are required and thus provide a $\Omega(1)$ lower bound.

$$C * C_0 = \Omega(1) \tag{3}$$

Where $C > 0$ and C_0 is the finite number of assignments and comparisons.

Since we are thinking of the best case and the best case returns after locating the key k, this is a constant operation that does not depend on the size of n. Since the best case has only a finite number of constant operation that is not dependent on the size of n, we have $0\big(1\big)$.

$$C_0 = 0\big(1\big) \tag{4}$$
$$C_0 \leq C * (1) \tag{5}$$

Where $0 < C \leq C_0$.

Since we have both $\Omega(1)$ and $0\big(1\big)$, we get $\Theta(1)$.

(b) *Using Θ notation, what is the worst case running time as function of n?*
Since this is a search algorithm, the worst case is when we search the entire array and cannot find key k. The *while* loop dictates how many iterations i are executed. Since we have established that i is on the order of $\lg(n)$ and fails on the last execution, we know that the *while* loop can execute a minimum number of $\lg(n) + 1$ times. This shows that we have $\Omega(\lg(n))$.

$$\lg(n) + 1 = \Omega(\lg(n)) \tag{6}$$
$$\lg(n) + 1 \geq C \lg(n) \tag{7}$$

We can pick $0 < C < 1$ for $n > 0$ and conclude that the worst case scenario for this binary search algorithm is $\Omega(\lg(n))$.

Inside the loop, there are a finite number of comparison and assignment operation that is not dependent on the size of n. We have the if, $elseif$, $else$ statements and the maximum number of comparisons to execute is 2. We also note that the size of n_i goes to 0 as i increases.

$$n_i = \frac{n}{2^i} = 0 \;\; for \;\; i \geq \lg(n) \tag{8}$$

This is a sure way of knowing that the function will terminate and terminate at the end of at most $\lg(n)$ iterations. In this sense we can define the maximum running time to be $0\big(\lg(n)\big)$.

$$\lg(n) + 1 = 0\big(\lg(n)\big) \tag{9}$$
$$\lg(n) + 1 \leq C \lg(n) \tag{10}$$

We can choose $C > 1$ for $n > 0$ to see that the maximum running time of this algorithm is $0\big(\lg(n)\big)$.

Since we have both $\Omega(\lg(n))$ and $0\big(\lg(n)\big)$, we have that the running time in the worst case scenario is $\Theta(\lg(n))$.

2. *Use this version of the algorithm to sort the following functions by order of asymptotic growth such that the final arrangement of functions $g1, g2, ..., g12$ satisfies the ordering constraint $g1 = \Omega(g2),\ g2 = \Omega(g3), \ldots, g11 = \Omega(g12)$.*

   (a) (**20 pts extra credit**) For each branch of the recursion tree, identify the chosen pivot element; for each level of the recursion tree and after all pivot elements at that level have been moved into their final location, give the global array ordering. (Hint: it may be easier to work out the final ordering by hand and then backtrack through the Quicksort operations to produce the recursion tree.)

   Starting with:

$$n, n^2, \sqrt{(2)}^{\lg n}, 2^{\lg^* n}, n!, (\lg n)!, \left(\frac{3}{2}\right)^n, n^{1/\lg n}, \lg(n!), e^n, 1$$

   i. Level: root. For our first pass through quicksort, partition choose 1 as the pivot element. Since every function is lower bounded by 1, we swap 1 with 1 and return 1. After the first pass nothing has changed.

$$n, n^2, \sqrt{(2)}^{\lg n}, 2^{\lg^* n}, n!, (\lg n)!, \left(\frac{3}{2}\right)^n, n^{1/\lg n}, \lg(n!), e^n, 1$$

   ii. Level: 1. The first left substep is now going to be the entire array excluding 1. Partition chooses e^n as our pivot. After this step we go from:

$$n, n^2, \sqrt{(2)}^{\lg n}, 2^{\lg^* n}, n!, (\lg n)!, \left(\frac{3}{2}\right)^n, n^{1/\lg n}, \lg(n!), e^n$$

   to

$$n!, e^n, \sqrt{(2)}^{\lg n}, 2^{\lg^* n}, n, (\lg n)!, \left(\frac{3}{2}\right)^n, n^{1/\lg n}, \lg(n!), n^2$$

   iii. First right subarray, is going to be zero elements since we start from the element after 1, which does't exist, and then go to the end.
   Globally, we have:

$$n!, e^n, \sqrt{(2)}^{\lg n}, 2^{\lg^* n}, n, (\lg n)!, \left(\frac{3}{2}\right)^n, n^{1/\lg n}, \lg(n!), n^2, 1$$

   iv. Level: 2 The next left subarray is just the element $n!$ so we can stop.

v. The right subarray are the elements after e^n to n^2:

$$\sqrt{(2)}^{\lg n}, 2^{\lg^* n}, n, (\lg n)!, \left(\frac{3}{2}\right)^n, n^{1/\lg n}, \lg(n!), n^2$$

Our pivot is n^2 and we transform this to:

$$(\lg n)!, \left(\frac{3}{2}\right)^n, n^2, \sqrt{2}^{\lg n}, 2^{\lg^* n}, n^{1/\lg n}, n\lg n, lg(n!), n$$

Globally we have:

$$n!, e^n, (\lg n)!, \left(\frac{3}{2}\right)^n, n^2, \sqrt{2}^{\lg n}, 2^{\lg^* n}, n^{1/\lg n}, n\lg n, lg(n!), n, 1$$

vi. Next left level we have:

$$(\lg n)!, \left(\frac{3}{2}\right)^n$$

with pivot $\left(\frac{3}{2}\right)^n$. This changes to:

$$\left(\frac{3}{2}\right)^n, (\lg n)!$$

This does not change any further.

vii. On the right side we have:

$$\sqrt{2}^{\lg n}, 2^{\lg^* n}, n^{1/\lg n}, n\lg n, lg(n!), n$$

with pivot n. This changes to:

$$n\lg n, lg(n!), n, \sqrt{(2)}^{\lg n}, 2^{\lg^* n}, n^{1/\lg n}$$

Globally, we now have:

$$n!, e^n, \left(\frac{3}{2}\right)^n, (\lg n)!, n\lg n, \lg(n!), n, \sqrt{(2)}^{\lg n}, 2^{lg^* n}, n^{1/\lg n}, 1$$

viii. Next left level we have:

$$n\lg n, \lg(n!)$$

with pivot $\lg(n!)$. $n\lg n$ is bounded from below by $\lg(n!)$ but this only switches $n\lg n$'s position with itself, and then the pivot switches position with itself. So this left subarray remains unchanged to:

$$n\lg n, \lg(n!)$$

ix. On the right side we have:

$$\sqrt{(2)}^{\lg n}, 2^{\lg^* n}, n^{1/\lg n}$$

with pivot $n^{1/\lg n}$. Both of our values are lower bounded by our pivot so after partition we still have:

$$\sqrt{(2)}^{\lg n}, 2^{\lg^* n}, n^{1/\lg n}$$

Globally, we still have:

$$n!, e^n, \left(\frac{3}{2}\right)^n, (\lg n)!, n \lg n, \lg(n!), n, \sqrt{(2)}^{\lg n}, 2^{lg^* n}, n^{1/\lg n}, 1$$

x. Next left level we have:

$$\sqrt{(2)}^{\lg n}, 2^{\lg^* n}$$

with pivot $2^{\lg^* n}$. This remains unchanged after partition. There is no further sorting of these as the next step are single, already sorted elements.

xi. Our right subarray is a trivial, 0 element array so this is also sorted.

Our final global array is still:

$$n!, e^n, \left(\frac{3}{2}\right)^n, (\lg n)!, n \lg n, \lg(n!), n, \sqrt{(2)}^{\lg n}, 2^{lg^* n}, n^{1/\lg n}, 1$$

(b) (30 pts) Give the final sorted list and identify which pair(s) functions $f(n), g(n)$, if any, are in the same equivalence class, i.e., $f(n) = \Theta(g(n))$.

$$(1)\ \lim_{n\to\infty}\frac{n!}{e^n}=\infty \qquad (7)\ \lim_{n\to\infty}\frac{\lg(n!)}{n}=\infty$$

$$(2)\ \lim_{n\to\infty}\frac{e^n}{\left(\frac{3}{2}\right)^n}=\infty \qquad (8)\ \lim_{n\to\infty}\frac{n}{\sqrt{2}^{\lg n}}=\infty$$

$$(3)\ \lim_{n\to\infty}\frac{\left(\frac{3}{2}\right)^n}{(\lg n)!}=\infty \qquad (9)\ \lim_{n\to\infty}\frac{\sqrt{2}^{\lg n}}{2^{\lg^* n}}=\infty$$

$$(4)\ \lim_{n\to\infty}\frac{(\lg n)!}{n^2}=\infty \qquad (10)\ \lim_{n\to\infty}\frac{2^{lg^* n}}{n^{1/\lg n}}=\infty$$

$$(5)\ \lim_{n\to\infty}\frac{n^2}{n\lg n}=\infty \qquad (11)\ \lim_{n\to\infty}\frac{n^{1/\lg n}}{1}=2$$

$$(6)\ \lim_{n\to\infty}\frac{n\lg n}{\lg(n!)}=1$$

These relations produces the list:

$\left|\ n!\ \right|\ e^n\ \left|\ \left(\frac{3}{2}\right)^n\ \right|\ (\lg n)!\ \left|\ n^2\ \right|\ n\lg n\ \left|\ \lg(n!)\ \right|\ n\ \left|\ \sqrt{2}^{\lg n}\ \right|\ 2^{\lg^* n}\ \left|\ n^{\frac{1}{\lg n}}\ \right|\ 1\ \left|\right.$

with $n\lg n$ in the same equivalence class as $\lg(n!)$ and $n^{\frac{1}{\lg n}}$ in the same equivalence class as 1.

3. *Consider the recursive function f, which takes an integer as an argument and returns some other integer.*

   (a) Write down the recurrence relation for the *value* returned by $f(n)$. Identify the base cases.

   Since we are only concerned with how the value is constructed, we restrict our view of $f(n)$ to the *else* statement. In this case, there are two recursive calls, two assignments, two multiplication, one subtraction, and two recursive calls. We can use the form of the function to develop a recurrence relation. $Value$ is created recursively. We can concentrate on the recursion and develop the following relationship.

$$a_n = 3 * a_{n-1} - 2 * a_{n-2} \tag{11}$$

   We can see from the if and $elseif$ statements of the function that the bases are $a_0 = 3$ and $a_1 = 5$.

   (b) Using the method of the *characteristic polynomial*, and solve the recurrence relation exactly.

   We construct the characteristic polynomial by noting that this recurrence relation is of degree 2. Labeling a_n as r^2 and moving all of the terms to the left side, we get

$$a_n = 3 * a_{n-1} - 2 * a_{n-2} \tag{12}$$
$$r^2 - 3r + 2 = 0 \tag{13}$$
$$(r-1)(r-2) = 0 \tag{14}$$

   We see that the roots of the characteristic polynomial are $r = 1$ and $r = 2$. We can use these to build our recurrence relation solution.

$$a_n = A(1) + B(2^n) \tag{15}$$

   We use our initial conditions (base cases) to find constants A and B.

$$a_0 = 3 = A + B(2^0) = A + B \tag{16}$$
$$a_1 = 5 = A + B(2^1) = A + 2B \tag{17}$$

Now we can solve for the constants A, B.

$$3 - B = A \tag{18}$$
$$\tag{19}$$

Now we substitute for A.

$$5 = (3 - B) + 2B \tag{20}$$
$$5 = 3 + B \tag{21}$$
$$2 = B \tag{22}$$

Using our value for B in one of the initial condition equations.

$$3 - 2 = A \tag{23}$$
$$1 = A \tag{24}$$
$$\tag{25}$$

Now we can build the explicit solution using these constants and the roots of the characteristic equation.

$$a_n = A(1) + B(2^n) \tag{26}$$
$$= 1 + 2(2^n) \tag{27}$$
$$= 1 + 2^{n+1} \tag{28}$$

(c) Show (prove) by induction that your solution is correct.

We can use $a_n = 3a_{n-1} - 2a_{n-2}$ as our recurrence relation, $a_n = 1 + 2^{n+1}$ as our solution, and note that our base cases are $a_0 = 3$ and $a_1 = 5$.

Base Step:
Plug in $n = 1$

$a_1 = 1 + 2^{1+1} = 1 + 4 = 5$

Our base case has been satisfied.

Inductive Step:
We want to show that $a_{n+1} = 1 + 2^{n+2}$

$$\begin{aligned} a_{n+1} &= 3a_n - 2a_{n-1} && (29)\\ &= 3(1 + 2^{n+1}) - 2(1 + 2^n && (30)\\ &= 3 + 3(2^{n+1}) - 2 - 2(2^n) && (31)\\ &= 1 + 3(2^{n+1}) - 2^{n+1} && (32)\\ &= 1 + 2(2^{n+1}) && (33)\\ &= 1 + 2^{n+2} && (34) \end{aligned}$$

We have shown by induction that our explicit solution to the recurrence relation is correct.

(d) Under the RAM model of computation, write down the recurrence relation for the *running time* of $f(n)$.

The only operations that are non-constant in this function are the recursion calls in the else statement. We can think of a recursion tree that branches into two sub-trees, $(n-1)$ and $(n-2)$. Since each node has two children, there are $2^n - 1 = O(2^n)$ nodes that have constant and finite atomic operations. With $n = 1$, there is only $2^1 - 1 = 1$ node and this makes sense since this is the root node. With $n = 2$, we have $2^2 - 1 = 3$ and this makes sense since there is the root node plus two child nodes. Our recurrence relation becomes

$$T(n) = T(n-1) + T(n-2) \tag{35}$$

where the recursion calls incur a constant cost and at the end of the call, the base cases are found which are also constant operations.

(e) Give a succinct explanation in terms of the shape of the recursion tree of why $O(2^n)$ is a trivial upper bound on the running time.

In terms of the recursion tree, we can learn about the shape by looking at each of the recursive calls. We have the $T(n-1)$ case and the $T(n-2)$ case. Let's assume that the left sub-tree is the $(n-1)$ case and the right sub-tree is the $(n-2)$ case. We can see that since the right sub-tree decreases by two instead of one, as the left sub-tree does, the recursion will reach a base case earlier. The depth of the left sub-tree is n since we are decrementing by one each call while the depth of the right sub-tree will be $n/2$ since it ends half as fast.

With each of the sub-trees changing at different rates, we get an unbalanced tree. $O(2^n)$ is an overestimate because having 2^n implies that the tree is balanced. We have shown that this recursion tree is unbalanced and the algorithm will perform much better.

(f) Using the method of characteristic polynomials, solve the time recurrence relation for a tight upper bound.

We can use the recurrence relation $T(n) = T(n-1) + T(n-2)$ and the base cases $T(0) = 2, T(1) = 3$.

We can think of the recursion in terms of sequences.

$$a_n = a_{n-1} + a_{n-2} \tag{36}$$

We can form the characteristic equation

$$r^2 = r + 1 \tag{37}$$

$$r^2 - r - 1 = 0 \tag{38}$$

Solving for r, we get $r = (1 + \sqrt{5})/2$ and $r = (1 - \sqrt{5})/2$

Since we have roots of multiplicity one, the format of the solution becomes of the form

$$a_n = A(\frac{1}{2}(1 + \sqrt{5}))^n + B(\frac{1}{2}(1 - \sqrt{5}))^n. \tag{39}$$

We can now utilize our initial conditions to find constants A and B.

Our base cases are when $n = 0$ and $n = 1$. For $n = 0$, we can count the if condition and the $return$ as 2 total atomic operations. For $n = 1$, we can add the extra $elseif$ comparison to get 3 atomic operations.

$$a_n = A(\frac{1}{2}(1+\sqrt{5}))^n + B(\frac{1}{2}(1-\sqrt{5}))^n \tag{40}$$

$$a_0 = 2 = A(\frac{1}{2}(1+\sqrt{5})^0 + B(\frac{1}{2}(1-\sqrt{5}))^0 = A + B \tag{41}$$

$$a_1 = 3 = A(\frac{1}{2}(1+\sqrt{5}))^1 + B(\frac{1}{2}(1-\sqrt{5}))^1 = \frac{A}{2}(1+\sqrt{5}) + \frac{B}{2}(1-\sqrt{5}) \tag{42}$$

$$\tag{43}$$

We can solve for A and B using substitution.

$$2 = A + B \tag{44}$$

$$2 - B = A \tag{45}$$

$$3 = \frac{A}{2}(1+\sqrt{5}) + \frac{B}{2}(1-\sqrt{5}) \tag{46}$$

$$6 = A(1+\sqrt{5}) + B(1-\sqrt{5}) \tag{47}$$

Substituting for A

$$6 = (2-B)(1+\sqrt{5}) + B(1-\sqrt{5}) \tag{48}$$

$$6 = 2 + 2\sqrt{5} - B - B\sqrt{5} + B - B\sqrt{5} \tag{49}$$

$$4 = 2\sqrt{5} - 2B\sqrt{5} \tag{50}$$

$$2 = \sqrt{5} - B\sqrt{5} \tag{51}$$

$$\frac{2}{\sqrt{5}} = 1 - B \tag{52}$$

$$B = 1 - \frac{2}{\sqrt{5}} \tag{53}$$

Substituting back for B to get A

$$A = 2 - (1 - \frac{2}{\sqrt{5}}) \tag{54}$$

$$A = 1 + \frac{2}{\sqrt{5}} \tag{55}$$

We get $T(n) = (1 + \frac{2}{\sqrt{5}})(\frac{1+\sqrt{5}}{2})^n + (1 - \frac{2}{\sqrt{5}})(\frac{1-\sqrt{5}}{2})^n$. This is a much tighter bound that 2^n.

4. *Solve the following recurrence relations using the method specified. Show your work.*

   (a) $T(n) = T(n-1) + n$ by "unrolling" (tail recursion)

   We can start "unrolling" the equation by computing
   $T(n-1) = T(n-2) + n - 1$ and working our way toward $T(0)$ which is the base case of the recursion and can be evaluated on the order of $\Theta(1)$.

$$\begin{aligned} T(n) &= T(n-1) + n && (56)\\ &= T(n-2) + n - 1 + n && (57)\\ &= T(n-3) + n - 2 + n - 1 + n && (58)\\ &\vdots && (59)\\ &= T(0) + 1 + 2 + ... + n - 1 + n && (60) \end{aligned}$$

   We end up with the sum

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2} \tag{61}$$

   We can say that the asymptotic behavior of this recurrence relation is $T(n) = \Theta n^2$ and always executes at $\Theta(n^2)$ since the sum always evaluates entirely as $T(n)$ is called.

   (b) $T(n) = 2T(n-2) + n^3$ by the Master method.

   With the Master theorem, we have 3 cases to consider. First, we designate $a = 2$, $b = 2$, $f(n) = n^3$. We can calculate $n^{\log_b a}$ and analyze its result among the cases.

$$\begin{aligned} n^{\log_b a} &= n^{\log_2 2} && (62)\\ &= n && (63) \end{aligned}$$

   In this case, we see that $n^{\log_b a} < f(n)$ and $n^3 = \Omega(n^{1+\epsilon})$ where $\epsilon = 2$. We can recognize this is the third case of the Master theorem. We now show one of the requirements of this case, namely $af(n/b) \leq cf(n)$ for some constant $c < 1$.

$$af(n/b) \leq cf(n) \tag{64}$$
$$=2(n/2)^3 \leq cn^3 \tag{65}$$
$$=2(n^3/8) \leq cn^3 \tag{66}$$
$$=n^3/4 \leq cn^3 \tag{67}$$

We can see that the statement is true for $c \geq 1/4$. Thus, we can conclude by the Master theorem that

$$T(n) = 2T(n-2) + n^3 = \Theta(n^3) \tag{68}$$

5. *Professor Septima Vector thinks she has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three (possibly empty) sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Professor Vector claims that any three keys $a \in A$, $b \in B$ and $c \in C$ must satisfy $a \leq b \leq c$. Prove that Vector's claim is false by giving a counterexample that is the smallest possible.*

   By way of contradiction, assume that any three keys $a \in A$, $b \in B$ and $c \in C$ must satisfy $a \leq b \leq c$.

   Let the values 42, 21, 20, and 23 be inserted into a valid binary search tree data structure in that order. The root node of the tree becomes 42 and it has 21 as its left child. 20 becomes the left child of 21 and 23 becomes the right child of 21.

   Assume the key we wish to find is 20. The search path contains the keys 42, 21, and 20. These keys make up the set B. A is the empty set since there are no keys to the left of the search path and C contains 23 since it is to the right of a key on the search path. We can choose a key b in set B and compare it with a key c in C. Let $b = 42$ since the root is included in the search path and let $c = 23$. In this way, these keys must satisfy $a \leq b \leq c$. Since A is the empty set, we can focus on $b \leq c$.

   $42 \leq 23$

   This is a contradiction since $42 > 23$. We have shown that this property if `false`.