COP 3502 - ~~10/8~~ 11 /2021

① General Alg Analysis (Q1 on Sec 2A of final exam)

② Recurrence Relations

```
Towers(n) {
    Towers(n-1);
    print
    Towers(n-1);
}
```

Let $T(n)$ = # steps/moves to solve towers of Hanoi puzzle with n disks.

$T(n) = T(n-1) + 1 + T(n-1)$, $T(0) = 0$

$T(n-1) = 2T(n-2)+1$

$T(n-2) = 2T(n-3)+1$

$$
\begin{aligned}
T(n) &= \boxed{2T(n-1)+1} \\
&= 2[2T(n-2)+1]+1 \\
&= 4T(n-2)+2+1 \\
&= \boxed{4T(n-2)+3} \\
&= 4[2T(n-3)+1]+3 \\
&= 8T(n-3)+4+3 \\
&= \boxed{8T(n-3)+7}
\end{aligned}
$$

After k steps: $2^kT(n-k)+(2^k-1)$, let $k=n$ and substitute

$$
\begin{aligned}
&= 2^nT(n-n)+(2^n-1) \\
&= 2^n \cdot 0 + 2^n - 1 \\
&= \boxed{2^n-1}
\end{aligned}
$$

Name: ____________________
UCFID: ____________________
NID: ____________________

1) (5 pts) ANL (Algorithm Analysis)

Given an array, `vals`, of size n, one can determine the sum of the elements in the array from index i through index j ($i \le j$), inclusive, simply by running a for loop through the elements:

```
int sum = 0;
for (int z=i; z<=j; z++)
    sum += vals[z];
```

This type of sum is known as a contiguous subsequence sum.

Note: There are more efficient ways to do this if many sums of this format need to be determined, but for the purposes of this problem, assume that this is how such a sum is determined.

(a) (3 pts) What is the worst case run time of answering **q** questions about contiguous subsequence sums on an array of size **n**? Express your answer in Big-Oh notation, in terms of both **n** and **q**. Give a brief justification for your answer.

for a single query worst case is $i=0$, $j=n-1$ and an $O(n)$ run-time.
To answer q queries worst case run time is $\boxed{O(nq).}$

(b) (2 pts) What is the best case run time of answering **q** questions about contiguous subsequence sums on an array of size **n**? Express your answer in Big-Oh notation, in terms of both **n** and **q**. Give a brief justification for your answer.

Best case single query is $i=j$, so $O(1)$ for one query.
Best case q queries is $\boxed{O(q).}$

1) (5 pts) ANL (Algorithm Analysis)

What is the best and worst case runtime for the following algorithm, in terms of the input parameter n? Give a brief explanation for your answers.

```
int foo(int * arr, int n){

   if (n == 0)
      return 0;

   int j = 0, i;

   for (i = 0; i < n; i++)
      if (arr[i] > arr[j])
         j = i;

   int nLen = n - j - 1;
   return arr[j] + foo(arr + j + 1, nLen);
}
```

Best case: $j = n-1$ on 1st run and we only do the for loop once. It took $\boxed{O(n)}$ time.

Worst case: $j = 0$ every time,
we loop n times
we loop n-1 times
we loop n-2 times

rec calls

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} = \boxed{O(n^2)}$$

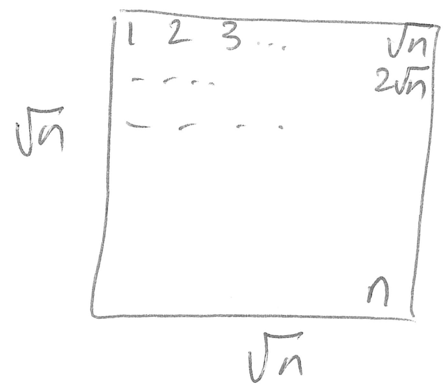
n boxes

1st k have gold coins

Ask: Does box x have a gold coin?

Only allowed to get 2 no answers.

What's the best strategy to determine k?



Ask about boxes $\sqrt{n}, 2\sqrt{n}, 3\sqrt{n}, \ldots$ until you get your 1st no.

Let's say $m\sqrt{n} = \text{yes}$

$(m+1)\sqrt{n} = \text{no}.$

Then ask in order $m\sqrt{n}+1, m\sqrt{n}+2, \ldots$

$$\text{Max \# questions} \leq \sqrt{n} + \sqrt{n} = O(\sqrt{n})$$

"Square Root Decomposition"

Name: ____________________
UCFID: ____________________
NID: ____________________

1) (10 pts) ANL (Algorithm Analysis)

There is a very long corridor of rooms, labeled 1 through n, from left to right. It is reputed that in the very last room, room *n*, there is the Treasure of the Golden Knight. Unfortunately, you don't know what *n* is equal to. Whenever you are in a particular room, you are allowed to ask questions of the form, "Is there a room 2^k slots to the right of my current location?", where k is a non-negative integer. For a fee, Knightro, an omnipresent, omnipotent, omniscient knight, will answer your question correctly, with either "yes" or "no." After you ask 1 or more questions from a single room, Knightro will move you, for free, to any of the rooms you asked a question about for which he replied "yes." Your goal is to get to room n by asking as few questions as possible, to reduce the fee that you pay Knightro. Devise a strategy to find the value of n and clearly outline this strategy. How many questions, in terms of n, will your strategy use, in the worst case? Answer, with proof, this last question with a Big-Oh bound in terms of n. **(Note: Any strategy that works will be given some credit. The amount of credit given will be based on how efficient your strategy is, in relation to the intended solution.)**

Keep asking question form $2^0, 2^1, 2^2, \ldots$ until you get your first no. Then you know you have to move less than 2^k slots where you got an answer of no for 2^k. Then in succession ask if you can move 2^{k-1} slots, 2^{k-2} slots etc. all the way back down to 2^0. Since n has some binary representation, this strategy will advance the correct # of spots. If n is inbetween 2^{k-1} and 2^k, we ask no more than 2k questions.

Roughly,

$n = 2^k$

$\log_2 n = \log_2 2^k$

$\log_2 n = k$

\# questions is $2k$

$\boxed{O(\lg n)}$

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

sidebar for fun

Binary Search

```
binsearch (arr, low, high) {
   // bc
   if (          )
       binsearch (arr, low, mid-1)
   else  binsearch(arr, mid+1, high)
}
```

Let $T(n)$ = run time of bin search on n elements

$$T(n) = \boxed{1 + T\left(\frac{n}{2}\right)}, \quad T(1) = O(1)$$

$$T(n) = T\left(\tfrac{n}{2}\right) + 1$$
$$= \left[T\left(\tfrac{n}{4}\right) + 1\right] + 1$$
$$= \boxed{T\left(\tfrac{n}{4}\right) + 2}$$
$$= \left[T\left(\tfrac{n}{8}\right) + 1\right] + 2$$
$$= \boxed{T\left(\tfrac{n}{8}\right) + 3}$$

Plug in value of k s.t.

$$\frac{n}{2^k} = 1$$
$$n = 2^k$$
$$k = \log_2 n$$

After k steps $T(n) = T\left(\frac{n}{2^k}\right) + k$

$$T(n) = \underline{T(1)} + O(\lg n)$$
$$= O(1) + O(\lg n)$$
$$= \boxed{O(\lg n)}$$

New Recurrence Relation

$$T(n) = \boxed{2T(\tfrac{n}{2}) + O(n)}, \quad T(1) = O(1)$$

$$= 2\left[2T(\tfrac{n}{4}) + O(\tfrac{n}{2})\right] + O(n)$$

$$= 4T(\tfrac{n}{4}) + \underline{O(n)} + \underline{O(n)}$$

$$= \boxed{4T(\tfrac{n}{4}) + 2(O(n))}$$

$$= 4\left(2T(\tfrac{n}{8}) + O(\tfrac{n}{4})\right) + 2(O(n))$$

$$= 8T(\tfrac{n}{8}) + O(n) + 2(O(n))$$

$$= \boxed{8T(\tfrac{n}{8}) + \underline{3}(O(n))}$$

$T(\frac{n}{2}) = 2T(\frac{n}{4}) + O(\frac{n}{2})$

$T(\frac{n}{4}) = 2T(\frac{n}{8}) + O(\frac{n}{4})$

After k steps $= 2^k T\left(\frac{n}{2^k}\right) + k \cdot O(n)$

let $\frac{n}{2^k} = 1$, $n = 2^k$

$k = \log_2 n$

$$= n\,T(1) + (\log_2 n) \cdot O(n)$$

$$= \underline{n} + \underline{O(n \lg n)}$$

$$= \boxed{O(n \lg n)}$$

fast fourier trans.

merge sort

1) (10 pts) ANL (Algorithm Analysis)

Consider the following problem: You are given a set of weights, $\{w_0, w_1, w_2, \ldots, w_{n-1}\}$ and a target weight T. The target weight is placed on one side of a balance scale. The problem is to determine if there exists a way to use some subset of the weights to add on either side of the balance so that the scale will perfectly balance or not. For example, if T = 12 and the set of weights was {6, 2, 19, 1}, then one possible solution would be to place the weights 6 and 1 on the same side of the balance as 12 and place the weight 19 on the other side.

Below is a function that solves this problem recursively, with a wrapper function to make the initial recursive call. In terms of n, the size of the input array of weights, **with proof,** determine the **worst case** run time of the wrapper function. (**Note: Since only the run time must be analyzed, it's not necessary to fully understand WHY the solution works. Rather, the analysis can be done just by looking at the structure of the code.**)

```
int makeBalance(int weights[], int n, int target) {
    return makeBalanceRec(weights, n, 0, target);
}

int makeBalanceRec(int weights[], int n, int k, int target) {
    if (k == n) return target == 0;
    int left = makeBalanceRec(weights, n, k+1, target-weights[k]);
    if (left) return 1;
    int right = makeBalanceRec(weights, n, k+1, target+weights[k]);
    if (right) return 1;
    return makeBalanceRec(weights, n, k+1, target);
}
```

$O(1)$ · $T(n-1)$ · $O(1)$ · $T(n-1)$ · $O(1)$ · $T(n-1)$

worst case

$$T(n) = 3T(n-1) + O(1)$$
$$= 3[3T(n-2) + O(1)] + O(1)$$
$$= 9T(n-2) + (3+1)O(1)$$
$$= 9[3T(n-3) + 1] + 3 + 1$$
$$= 27T(n-3) + (9+3+1)O(1)$$

k steps $\quad = 3^k T(n-k) + \sum_{i=0}^{k-1} 3^i \qquad T(1)=1$

plug in $n-k=1$, $k=n-1$ $\quad = 3^{n-1} T(1) + \sum_{i=0}^{n-2} 3^i \qquad \boxed{O(3^n)}$

$$= 3^{n-1} + 3^{n-2} + 3^{n-3} + \cdots + 1 = \frac{3^n - 1}{3-1}$$