COP 3502 10/19/23

Fall 2023

Consider the following problem:

Given two input values, ***n*** and ***k***, determine the number of strings of length ***n***, which only contains A's and B's, that have a run of ***k*** or more consecutive B's.

One algorithm to solve the problem is as follows:

Recursively generate each possible string of ***n*** letters, each A's and B's. These can be generated in alphabetical order, never storing more than 1 of the strings at the same time.

For each string generated, loop through the string from left to right, keeping a running tally of the current number of B's. (For example, with the string ABBABBBAAB, the running counter would update as follows 0 → 1 → 2 → 0 → 1 → 2 → 3 → 0 → 0 → 1.) If this running tally ever equals or exceeds ***k***, add 1 to a global counter storing the final result. For simplicities sake, assume that the loop completes going through the whole string before 1 is potentially added to the global counter.

With proof, determine the Big-Oh runtime of this algorithm in terms of the input parameter, ***n***.

Summer 2023

What is the Big-Oh memory usage for the function call `createNode(N)`? Please provide your answer in terms of the input parameter, N. **Please justify your answer by either evaluating an appropriate recurrence relation or summation.**

let T(N) = amt of mem created

```
typedef struct Node Node;
struct Node {
   Node ** children;
   int val;
};
Node * createNode(int N) {
1 *Node * res = (Node *) malloc(sizeof(Node));
   if (N == 0) return res;
2 *res->children = (Node **) malloc(sizeof(Node*) * N);
3 *res->children[0] = createNode(N / 2);
   res->val = 0;
   for (int i = 0; i < N; i++)
      res->val += i;

   return res;
}
```

$$T(N) = 1 + N + T\left(\frac{N}{2}\right)$$

$$T(N) = T\left(\frac{N}{2}\right) + O(N)$$

$$= cn + \frac{cn}{2} + \frac{cn}{4} + \dots c$$

$$= cn\left(1 + \frac{1}{2} + \frac{1}{4} + \dots\right) \le cn\sum_{i=0}^{\infty}\left(\frac{1}{2}\right)^i = 2nc = O(n)$$

Master Thm

$A=1,\ B=2,\ k=1$

$B^k > A \Rightarrow O(n)$

Summer 2022

What is the worst case Big-Oh runtime for the function **f**, in terms of its input parameter n? You may assume that the array pointed to by arr is of length n. (Grading note: 2 pts will be awarded for the answer, 8 pts for the proof of the answer. Your proof must include either summations or recurrence relations related to the code below.)

Let $T(n)$ = runtime of fHelp $n = high - low + 1$

```
int f(int* arr, int n, int minVal) {
    return fHelp(arr, 0, n-1, minVal);
}

int fHelp(int* arr, int low, int high, int minVal) {
    if (low > high) return 0;
    if (low == high) return arr[low] >= minVal;

    int mid = (low+high)/2;
    int left = fHelp(arr, low, mid, minVal);
    int right = fHelp(arr, mid+1, high, minVal);
    int res = left;
    if (right > left)
        res = right;

    int alt = 0, i;
    for (i=mid; i>=low; i--) {
        if (arr[i] < minVal) break;
        alt++;
    }
    for (i=mid+1; i<=high; i++) {
        if (arr[i] < minVal) break;
        alt++;
    }

    if (alt > res) res = alt;
    return res;
}
```

] Size n input

$T(\frac{n}{2})$

$T(\frac{n}{2})$

] Size $\frac{n}{2}$ input

Overall low to high

$O(n)$

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

] rest $O(1)$

Merge Sort $\Rightarrow O(n\lg n)$

Master Thm $A=2, B=2, k=1$ $\rightarrow O(n\lg n)$

Spring 2022

What is the best and worst case runtime for the following algorithm, in terms of the input parameter n? You may assume that the array pointed to by arr is of length n. Give a brief explanation for your answers.

```
int foo(int * arr, int n, int value){
    int cur = 0, jump = n/2;
    while (jump > 0) {
        if (value > arr[cur])
            cur += jump;
        else if (value == arr[cur])
            return cur;
        jump = jump/2;
    }

    return cur;
}
```

$n/2$

$n/4$

$n/8$

$\vdots$

1

] repeated div by 2

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

$k = \log_2 n \rightarrow$ worst case $O(\lg n)$

Fall 2023

Gen all strings length n of As Bs

$n=3$ AAA, AAB, ... , BBB

for each string, we'll loop through it...

#strings $= 2^n$,

for each string we do: $O(n)$

$O(n2^n)$

Timing Prob

$O(n^2)$ 40 ms $n = 20{,}000$

time for $n = 70{,}000$

$$T(n) = cn^2$$

$$T(20000) = c(20000)^2 = 40 \text{ ms}$$

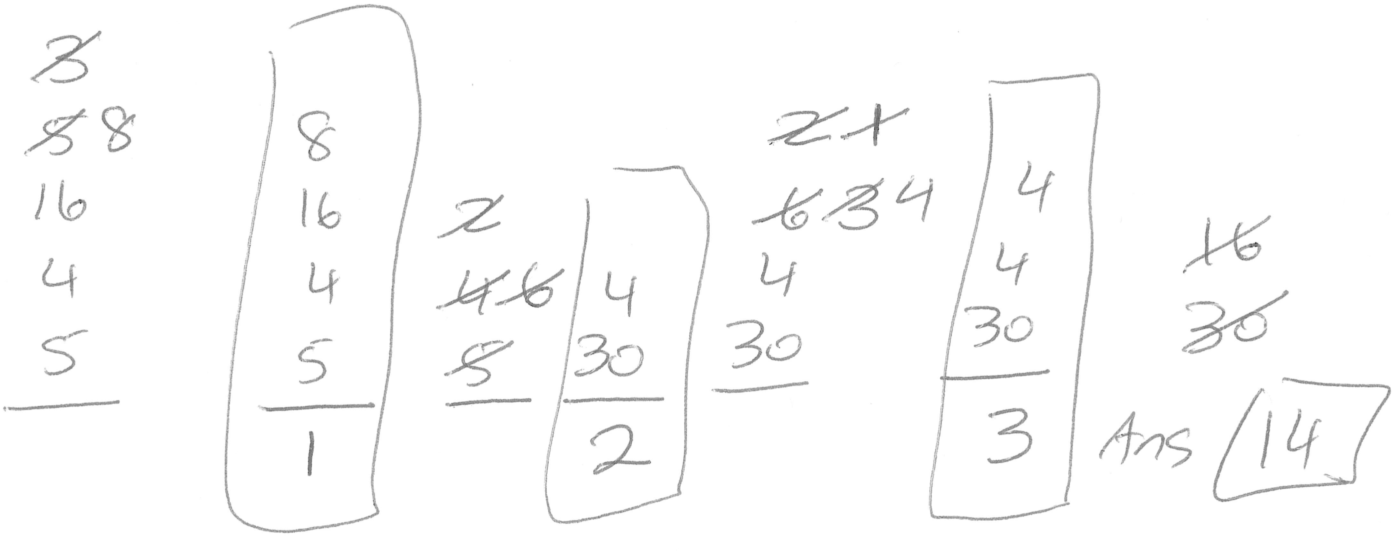
$$c = \frac{40 \text{ ms}}{(2\times10^4)^2}$$

$$T(70000) = \frac{40\text{ms}}{(2\times10^4)^2} \times (7\times10^4)^2 = \frac{40\text{ms} \times 49}{4}$$

$$= \boxed{490 \text{ ms}}$$

Eval Postfix

5 4 16 5 3 + ^{1}/ + * 4^{2} 6 2 / 1 +^{3} * -



```
for (i=0; i<n; i++)
    for (j=0; j<i; j++)
        //const stuff
```

0, 1, 2, ... n-1 times

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

```
sum=0
while (n>0) {
    for (i=0; i<n; i++)
        sum++;
    n = n/2;
}
```

$$n + \frac{n}{2} + \frac{n}{4} + \cdots$$

$$\le \sum_{i=0}^{\infty} n\left(\frac{1}{2}\right)^i$$

$$= n \times 2 = O(n)$$

```
sum=0
while (n>0) {
    sum++;
    n=n/3;
}
```

$$\frac{n}{3^k} = 1$$

$$k = \log_3 n$$

$$O(\lg n)$$