CS180

Derek Xu
704751588.

Problems 2,3,7,8

2 a) $t = 1\text{hr} = 3600\text{sec}$

$r = 10^{10}$ operation / sec

$\therefore r = 3.6 \cdot 10^{13}$ operation/hr

$n^2 = 3.6 \cdot 10^{13}$

$\boxed{n = 6 \cdot 10^6}$

b) $n^3 = 3.6 \cdot 10^{13}$

$n = 33019.27 \rightarrow \boxed{33019}$

c) $100n^2 = 3.6 \cdot 10^{13}$

$n^2 = 3.6 \cdot 10^{11}$

$\boxed{n = 6 \cdot 10^5}$

d) $n \log n = 3.6 \cdot 10^{13}$

$\log n^n = 3.6 \cdot 10^{13}$

By stirling's approx:

$\log n! = 3.6 \cdot 10^{13}$

assume log base 2

$n! = 2^{3.6 \cdot 10^{13}}$

???

TA said use Wolfram Alpha:

$n \log n = 3.6 \cdot 10^{13}$

$\boxed{n \approx 1.29095 \cdot 10^{12}}$

... assuming log base e

e) $2^n = 3.6 \cdot 10^{13}$

$n = \log_2(3.6 \cdot 10^{13})$

$= 45.033 \rightarrow \boxed{45}$

f) $2^{2^n} = 3.6 \cdot 10^{13}$

$n = \log_2(\log_2(3.6 \cdot 10^{12}))$

$= 5.4929 \rightarrow \boxed{5}$

3 In ascending order:

$f_2 = \sqrt{2n}$

$f_3 = n + 10$

} polynomial of deg ≤ 1

$f_6 = n^2 \log n$ } polynomial of deg 2 · logarithm

$f_1 = n^{2.5}$ } polynomial of deg ≥ 2

$f_4 = 10^n$

$f_5 = 100^n$

} exponential

7 Given: ea. line has $\leq c$ words

song has $= n$ words

encoded song has length $f(n)$.

Let encoding be as follows:

an array w/ index of line # and element of words in specified line #.

ex/

index	element
0	"A partridge in a pear tree"
1	"Two Turtle Doves"
2	"Three french horns"
⋮	⋮

The corresponding script to print the song is as follows:

Let a be the array of encoded song.

Let l be length of a

```
for i in range [0, l):
    for j in range [i, 0]:
        print a[j]
```

Note: total # of print statements (i.e. # of lines in song)

$$= 1+2+3+\dots+l = \frac{l(l+1)}{2} = \frac{l^2+l}{2}$$

in song

# of lines assuming lines are length αc words on avg

$$= \frac{n}{\alpha \cdot c}$$

$$\frac{n}{\alpha c} = \frac{l^2+l}{2} \geq \frac{l^2}{2} \rightarrow l \leq \sqrt{\frac{2}{\alpha c}} \cdot \sqrt{n}$$

Note: each line has αc words s.t. $c \ll n$.

$\therefore$ total # of words in encoding

$$= f(n) = \alpha c \sqrt{\frac{2}{\alpha c}} \cdot \sqrt{n} = \boxed{O(\sqrt{n})}$$

Le

8.a) Given: $k = 2$ jars

n = rungs on ladder

Solution:

Let the rungs on the ladder be labeled $1, 2, 3, \ldots, n$. Let the function $int(i)$ return $\lfloor i \rfloor$, where i is a non-integer value. Note, this is an $O(1)$ operation in most computer languages. Let // denote comment

Algorithm:

1 Let $i = 0$
2 Let $j = \sqrt{n}$
3 while (2 jars not broken & $i \leq n$)
4   Let $i = i + j$
5   Test the ith rung
6   // that is drop the jar at the ith rung.
7 Let $i = i - j$
8 while (1 jar not broken & $i \leq n$)
9   Let $i = i + 1$
10   Test the ith rung.
11 return $i - 1$

At a high level, this algorithm tests rungs in $\sqrt{n}$ intervals w/ the 1st jar & finds the exact rung w/ the 2nd jar.

<u>Correctness Proof</u>

Rewritten to improve readability:
Proof by Contradiction:

Define: a safe rung is a rung where if the jar is dropped it does not break
Assume (i-1)th rung is not the highest safe rung → 2 cases

Case 1: ith rung is safe
This is false as it caused the program to leave the while loop on line 8, meaning that this rung either is greater than n (does not exist on the ladder), or it is not safe.

Case 2: (i-1)th rung is not safe
This is false as the while loop on line 8 exited on the ith rung. If it were true, the while loop would have exited n the (i-1)th rung and returned the (i-2)th rung.

Therefore, (i-1)th rung is highest safe rung!

Time Complexity

The following lines are $O(1)$: 1, 2, 7, 11

Denote them w/ c_1

The following lines are in while loop on line 3 and $O(1)$:

Denote them w/ c_2

The following lines are in while loop on line 8 and $O(1)$:

Denote them w/ c_3

Observe first while loop:

we iterate through at most n rungs at $\sqrt{n}$ rungs at a time.

$\therefore$ at most $n/\sqrt{n} = \sqrt{n}$ loops

Observe second while loop:

we know that loop will exit between i & $i+\sqrt{n}$ b/c i is a safe rung & $i+\sqrt{n}$ is not safe. we iterate 1 rung at a time $\therefore$ at most $\sqrt{n}/1 = \sqrt{n}$ loops

Total run time is:

$c_1 + c_2\sqrt{n} + c_3\sqrt{n}$

$= c_1 + (c_2 + c_3)\sqrt{n}$

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

Note $\lim_{n \to \infty} \frac{\sqrt{n}}{n} = 0$

b) Given: $k > 2$ jugs

n = rungs on ladder

We will define everything as we did in part a). Assume $k \ll n$; if $k \approx n$, we can use the binary search algorithm.

Algorithm:

```
1  Let i = 0
2  Let j = k-1
3  Let r = n^{j/k}
4  while (j ≥ 0)
5     while ((j+1) jars not broken & i ≤ n)
6         Let i = i + r
7         Test the i^th rung.
8     Let i = i - r.
9     Let j = j - 1
10    Let r = n^{j/k}
11 return i
```

At a high level, this algorithm tests rungs in intervals of $n^{j/k}$ w/ $j \in \{k-1, k-2, \dots, 0\}$ notice if we let $k=2$, we get the same algorithm from part a). Notice if we remove the constraint $k << n$, there are some values that make the algorithm slow, to counteract this, set interval, r, to $\min(n^{j/k}, (1/2)^{k-j} n)$ on lines 10 and 3. This effectively makes the algorithm binary search w/ time $O(\log n)$.

Correctness Proof

Rewritten to improve readability:
Proof by Contradiction:

Note: On the last iteration of the while loop on line 4, j=0 → r=n^j/k=1
Define: a safe rung is a rung where if the jar is dropped it does not break

Assume ith rung is not the highest safe rung → 2 cases

Case 1: (i+1)th rung is safe
This is false as it caused the program leave the while loop on line 5, meaning that this rung either is >n (does not exist on the ladder), or it is not safe. This is because the (i+1)th rung is the same as the (i+r)th rung since it exited on the last iteration of the while loop in line 4.

Case 2: ith rung is not safe
This is false as the while loop on line 5 exited on the (i+1)th rung. If it were true, the while loop would have exited on the ith rung and returned the (i-1)th rung.

Therefore, ith rung is highest safe rung!

Time Complexity

The following lines are $O(1)$: 1, 2, 3, 11

Denote them w/ c_1

The following lines are in the while loop on line 4 and not in the while loop on line 5 and $O(1)$: 8, 9, 10

Denote them w/ c_2

The following lines are in the while loop on line 5 and $O(1)$: 6, 7

Denote them w/ c_3

Observe first while loop:

we iterate j in range $[k-1, 0]$ at 1 rung at a time

$\therefore$ at most $(k-1+1)/1 = k$ loops

Observe second while loop:

For $r = n_j^{\alpha/k}$ we are left with $n^{(\alpha+1)/k}$ elements $\longleftarrow$ Claim

Prove by Induction:

Base case: $\alpha = k-1$,

we have n elements left $n = n^{k/k}$

Inductive Step:

Assume $n^{(\alpha+1)/k}$ elements in set,

Line 9 indicates that we subtract 1 from α $\therefore r = n^{\alpha/k}$

$\therefore$ we iterate through $n^{(\alpha+1)/k}$ rungs at $r = n^{\alpha/k}$ rungs at a time

$\therefore$ at most $n^{(\alpha+1)/k} / n^{\alpha/k} = n^{1/k}$ loops.

Total Run time is:

$c_1 + k(n^{1/k} c_3 + c_2)$ $\longleftarrow$ Note: as discussed earlier $k << n$ (otherwise use binary search w/ runtime $O(\log n)$) $\therefore$ we treat k as constant

$= kc_3 n^{1/k} + (c_1 + kc_2)$

$= \boxed{O(n^{1/k})}$

$$\lim_{n \to \infty} \frac{n^{1/k}}{n^{1/(k-1)}} = 0$$