Nome: Victor Akio Domoto RA: 150546 Turma: I

1)

2) Determinando o tempo de execução de cada algoritmo:

I. Seja $t(n)$ uma função do tempo para um problema de tamanho n, aplicando o teorema Mestre, temos:

$$t(n) = \begin{cases} \Theta(1) & n \le 1 \\ 5t(n/2) + \Theta(n) & n > 1 \end{cases} \Rightarrow a = 5,\ b = 2 \Rightarrow n^{\log_b a} = n^{\log_2 5} \text{ e } f(n) = n$$

• Portanto, comparando $n^{\log_2 5}$ com $f(n)$ temos o caso 1 do teorema em questão, de modo que $\varepsilon = 1$ e $n^{\log_2 5} > n^1$

$\therefore t(n) = \Theta(n^{\log_b a}) \Rightarrow t(n) = O(n^{\log_2 5})$.

II. $t(n) = \begin{cases} \Theta(1) & n \le 1 \\ 2t(n-1) + \Theta(1) & n > 1 \end{cases}$, analisando cada iteração i de $t(n)$ tal que $i \le n$, temos:

$i = 1 \quad t(n) = 2t(n-1) + 1$

$i = 2 \quad t(n) = 2(2t(n-2)+1)+1 = 2^2 t(n-2) + 2 + 1$

$i = 3 \quad t(n) = 2^2(2t(n-3)+1) + 2 + 1 = 2^3 t(n-3) + 2^2 + 2 + 1$

• Note que para cada iteração i, temos $2^i t(n-i) + 2^{i-1} + 2^{i-2} + \dots + 2^2 + 2 + 1$.

Observe que a sequência da direita se trata de uma potência de 2 $(2^i - 1)$.

Portanto, podemos escrever: $2^i t(n-i) + 2^i - 1$

• Se formos analisar para um $i \to n$, temos que:

$2^i t(n-i) + 2^i - 1 = 2^n t(n-n) + 2^n - 1$, de modo que $t(0) = \Theta(1)$, pela definição de t.

$\Rightarrow 2^n t(0) + 2^n - 1 = 2^n + 2^n - 1 = 2^{n+1} - 1 \Rightarrow O(2^n)$.

III. $t(n) = \begin{cases} \Theta(1) & n \le 1 \\ 9t(n/3) + \Theta(n^2) & n > 1 \end{cases}$, aplicando o teorema Mestre, temos:

$a = 9,\ b = 3 \Rightarrow n^{\log_b a} = n^{\log_3 9} = n^2$ e $f(n) = n^2$

• Comparando $n^{\log_b a}$ com $f(n)$ conclui-se que temos o caso 2 do teorema, tal que $n^{\log_b a} = f(n) \Rightarrow n^2 = n^2$.

Portanto, por definição, temos: $t(n) = O(n^{\log_b a} . \lg n) \Rightarrow t(n) = O(n^2 \lg n)$.

Para escolher um algoritmo, é necessário verificar qual é menor assintoticamente, $n^{\log_2 5}$, 2^n e $n^2 \lg_2 n$. Note que de imediato o algoritmo II, 2^n, tem um tempo de execução maior que os outros.

Portanto, basta analisar $n^{\log_2 5}$ e $n^2 \lg_2 n$, de modo que, a princípio, o algoritmo I para ser mais eficiente. Assim, basta provar a seguinte inequação: $n^{\log_2 5} < n^2 \lg_2 n$

$$\Rightarrow n^{\log_2 5} < n^2 \lg_2 n \Rightarrow \frac{(n^{\log_2 5} < n^2 \lg_2 n)}{n^2} = \frac{n^{\log_2 5}}{n^2} < \lg_2 n \Rightarrow \boxed{n^{\log_2 5 - 2} < \lg_2 n}$$

* Observe que $\log_2 5 - 2 \approx 0.32$

$\therefore n^{0.32} < \lg_2 n \Rightarrow \boxed{n > 2.55} \Rightarrow n^{\log_2 5} < n^2 \lg_2 n \;\; \forall n > 2.55$.

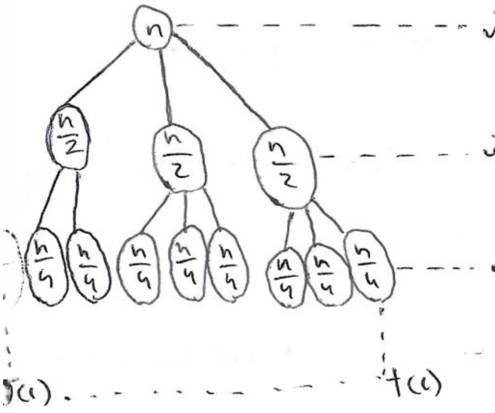
Entretanto, como fizemos uma aproximação de $n^{\log_2 5}$ para 0.32, temos que pegar o teto de n para garantir a inequação. $\Rightarrow \forall n > 3$, $n^{\log_2 5} < n^2 \lg_2 n$

Portanto, o algoritmo I é o escolhido por ser o mais eficiente. ∎

2) A)

$\Rightarrow t(n) = 3t(\lceil n/2 \rceil) + n$.

a) Pela árvore de recursão nós temos que a função acima diz que dado um problema de tamanho n, nós teremos 3 chamadas recursivas, de modo que cada uma delas terá $\lceil n/2 \rceil$. Além disso, ainda temos o termo n, que diz respeito ao tamanho da entrada.



$i = 0 \Rightarrow t(n) = n \Rightarrow t(n) = \frac{n}{2^0} = n$

$i = 1 \Rightarrow t(n) = \frac{n}{2} \Rightarrow t(n) = \frac{n}{2^1} = \frac{n}{2}$

$i = 2 \Rightarrow t(n) = \frac{n}{4} \Rightarrow t(n) = \frac{n}{2^2} = \frac{n}{4}$

Note que para cada iteração i, nós temos: $\frac{n}{2^i}$. Entretanto, essa sequência só irá terminar quando tivermos t(1).

$\Rightarrow$ A sequência só irá terminar quando $\frac{n}{2^i} = 1 \Rightarrow i = \log_2(n)$ (altura).

Portanto, o último nível será $\log_2 n$. Como $t(n)$ é dado pela soma de cada nível, temos: $\left(\sum_{i=0}^{\log_2 n} \frac{3^i}{2^i}\right) n \Rightarrow t(n) = \left(\cancel{\frac{3^0}{2^0}}\right)n + \left(\frac{3^1}{2^1}\right)n + \left(\frac{3^2}{2^2}\right)n + \dots + \left(\frac{3^{\log \frac{n}{2}}}{2^{\log \frac{n}{2}}}\right) n$

$$t(n) = n + \frac{3}{2} + \left[\left(\frac{3}{2}\right)^2\right]n + \dots + \left(\frac{3^{\log_2 n}}{2^{\log \frac{n}{2}}}\right) n$$

∴ Para $i \to \log \frac{n}{2} \Rightarrow t(1) = \frac{n}{2^i}$

$$t(n) = 3^i n/2^i + 3^{i-1} n/2^{i-1} + \dots + 3^1 n/2^1 + 3^0 n/2^0$$

$$= \frac{n((3/2)^i - 1)}{\frac{3}{2} - 1} = 2n((3/2)^{i+1} - 1) = 2n.((3/2).(3/2)^i - 1) = 2n\left(\cancel{\frac{3}{2}}\right)\left(\frac{3}{2}\right)^i - 2n$$

$= 3n\left(\frac{3}{2}\right)^i - 2n$. Note que $\frac{n}{2^i} = 1 \Rightarrow n = 2^i$

$$\Rightarrow t(n) = 3.(\cancel{2^i})\left(\cancel{\frac{3}{2}}\right)^i - 2(2^i) = 3.3^i - 2.2^i = 3.3^{\log_2 n} - 2.2^{\log \frac{n}{2}}$$

$$= \boxed{3n^{\log \frac{3}{2}} - 2n} \Rightarrow O(n^{\log \frac{3}{2}}) \blacksquare$$

2) B)

2) b) Pelo método da substituição, temos: $\begin{cases} 1 & , n = 1 \\ 3t(\lceil n/2 \rceil) + n, & n > 1 \end{cases}$

- Chutando que $t(n) \in O(n^{\log \frac{3}{2}})$, basta provar por indução que $t(n) \leq c.n^{\log \frac{3}{2}}$.

- Caso base: $n = 1$, por definição temos que $t(1) = 1$ e $c.1^{\log \frac{3}{2}} = 1.c = c$.

∴ o caso base é verdadeiro $\forall c \geq 1$.

Logo, provaremos que $t(n) \leq c.n^{\log \frac{3}{2}}$ se $n > 1$.

- Hipótese Indutiva: $t(k) \leq k^{\log \frac{3}{2}}.c \quad \forall \; 1 \leq k \leq n$

↳ Note que no item (a) encontramos que $t(n)$ pode ser escrito como uma série.

Portanto:

$$t(k) = \left[\sum_{i=0}^{\log_2 k} \left(\frac{3^i}{2^i}\right)\right] k \Rightarrow t(k) = k.\sum_{i=0}^{\log \frac{k}{2}} \left(\frac{3}{2}\right)^i$$

$\therefore$ Pela H.I: $T(K) \leq c.K^{\log\frac{3}{2}}$

$$\Rightarrow \underbrace{K.\sum_{i=0}^{\log_2 K}\left(\frac{3}{2}\right)^i}_{\text{Solução encontrada no item (a)}} \leq c.K^{\log\frac{3}{2}} \Rightarrow K.\sum_{i=0}^{\log_2 K}\left(\frac{3}{2}\right)^i = \left(3K^{\log\frac{3}{2}} . 2K\right)$$

$$\Rightarrow \left(3K^{\log\frac{3}{2}} - 2K\right) \leq c.K^{\log\frac{3}{2}} \Rightarrow c = \left(\frac{3K^{\log\frac{3}{2}} - 2K}{K^{\log\frac{3}{2}}}\right) \Rightarrow \boxed{c \geq 3 - \frac{2K}{K^{\log\frac{3}{2}}}}$$

• Observe que para um K suficientemente grande, temos:

$$\lim_{K\to\infty} 3 - \frac{2K}{K^{\log\frac{3}{2}}} \overset{L'H}{=} \lim_{K\to\infty} 3-0 = \boxed{3} \Rightarrow \forall c \geq 3,\ t(K) \leq c.K^{\log\frac{3}{2}} \Rightarrow O(n^{\log\frac{3}{2}})$$

3) A)

(3)

Problema: encontrar um subconjunto S de maior tamanho possível tal que a função f mapeia todo elemento de S a outro elemento de S ($f: S \to S$).

• Base: para um conjunto $A = \{a_1, a_2, a_3\}$, de modo que $f(a_1) = a_2$ e $f(a_2) = a_1$, temos que $S = \{a_1, a_2\}$. Portanto, para todo $n >= 2$ temos que é possível definir um subconjunto S tal que o mapeamento é dado por $f: S \to S$ e 1:1.

• Hipótese indutiva: para um $n >= 2$, teremos um conjunto $A = \{a_1, a_2, \dots, a_n\}$. Portanto, sendo f uma função 1:1, $f(A) = \{f(a_1), f(a_2), \dots, f(a_n)\}$.

Se $f(A[i])$ tiver um único correspondente e, além disso, for o correspondente de outro $f(A[J])$, tal que $A[i] != A[J]$, então temos um subconjunto S tal que $f: S \to S$.

• Passo: sabendo que o caso base está bem definido, podemos aplicar a H.I. em um conjunto A de qualquer tamanho $n >= 2$, uma vez que, se as condições forem satisfeitas, podemos garantir que no conjunto S teremos apenas $f: a \to a$ e 1:1.

3) B)

3) b) Dado um conjunto A de tamanho n>=2, temos um subconjunto S e uma função f de mapeamento, tal que f[A i] = aplica a função no elemento a_i.

```
mapeamento(A, n) {
  if(n==0) {
    if(F[A[n]]==A[i] && A[i]=F[A[j]]) {
        S[n] = A[n];
        return 0;
    }
  }else {
    if(F[A[i]]==A[i] && A[i]==F[A[j]]) {
        S[n]=A[n];
    }
  }
  mapeamento(A, n-1);
  return S;
}
```

■

3) C)

3) c) Complexidade em termos de n:

Dado o algoritmo do item (b), podemos definir a função t(n) como:

$$t(n) = \begin{cases} \Theta(1) & n=0 \\ t(n-1)+\Theta(1) & n>0 \end{cases}$$

- Analisando a cada iteração i:

$t(n) = t(n-1) + \Theta(1)$

$t(n) = t(n-2) + 2.\Theta(1)$

$t(n) = t(n-3) + 3.\Theta(1)$

$\vdots$

$T(n-i) + i\,\Theta(1)$

- Note que a cada iteração temos um padrão e ele só irá acabar quando i = n. Logo,

$t(n) = t(n-i) + i.\Theta(1)$

$i \to n$: $t(n) = t(\underbrace{\cancel{n-n}}_{\Theta(1)}) + n.\Theta(1)$

$i \to n$: $t(n) = n + \Theta(1) \Rightarrow t(n) \in O(n)$.

4) Quantidade ótima de tentativas: realizar a menor quantidade possível de tentativas.

A) Para conseguir uma ótima quantidade de tentativas, é possível realizar o teste como um algoritmo recursivo.

Note que não é necessário passar por todos os andares, basta dividirmos os testes para T(n/2) onde n é a quantidade de andares. Dessa forma, encontraremos em qual intervalo o celular quebra.

CASO I

Como a quantidade de celulares não é um problema, começar pelo 100º andar é um bom palpite, uma vez que se ele for falso, isto é, o celular não quebrar, então temos que o celular não quebra para nenhum outro andar.

Então seja f (n) uma função que representa a integridade do celular no nº andar, se f (100) = false, isto é, o celular não está quebrado após ser jogado do 100º andar, então temos a solução do problema, $\forall$ n $\leq$100 o celular não quebra;

CASO II

Assim como foi informado acima, nós queremos encontrar um intervalo no qual o celular quebra. Portanto, uma boa distribuição seria T(n/2) = n, n/2, n/4, n/8... 1. Entretanto, só teremos o CASO II se o I for verdadeiro, ou seja, o celular quebrou ao ser jogado do 100º andar, então podemos assumir esse fato. Assim, teríamos apenas testes a partir do n.

A cada chamada recursiva nós iremos verificar se o celular está quebrado. Caso esteja, isso implica que o limite de resistência do dispositivo está no intervalo $[n_i, n_{i-1}]$, onde n é a quantidade de andares e i a i-ésima iteração.

Como por exemplo, suponha que o primeiro teste, ou seja, (n/2) = (100/2) = 50º andar, o celular quebre. Isso implica que, seja X o andar exato no qual o celular quebra, então X está entre 50 e 100 $[n_i, n_{i-1}]$.

Caso o celular quebre em algum dos testes, temos, portanto, o intervalo $[n_i, n_{i-1}]$ de X, então basta ir aplicando novamente a chamada recursiva até que encontremos X. Portanto, teríamos T ($[n_i, n_{i-1}]$ / 2), T ($[n_i, n_{i-1}]$ / 4) ... T (1).

Então, vamos, por exemplo, supor que o celular quebre no 77º andar. Iremos dividindo o intervalo $[n_i, n_{i-1}]$ em 2 a cada tentativa:

- T(n/2) = 50º andar, f (50) = false (celular não quebra);
- => T ($[n_i, n_{i-1}]$ / 2) = T ([50, 100] / 2) = 75º, f (75) = false (celular não quebra);
- => T ($[n_i, n_{i-1}]$ / 4) = T ([75, 100] / 4) = 87º, f (87) = true (celular quebra);
- => T ($[n_i, n_{i-1}]$ / 8) = T ([75, 87] / 8) = 81º, f (81) = true (celular quebra);
- => T ($[n_i, n_{i-1}]$ / 16) = T ([75, 81] / 16) = 78º, f (78) = true (celular quebra);
- => T [$[n_i, n_{i-1}]$ / 32] = T ([75, 78] / 32) = 76º, f (76) = false (celular não quebra);
- => T ($[n_i, n_{i-1}]$ / 64) = T ([76, 78] / 64) = 77º, f (77) = true (celular quebra);
- Note que este foi o último intervalo possível, pois não existem andares, isto é, número inteiros, entre o 76º e 78º (T ([76, 78])) andar.

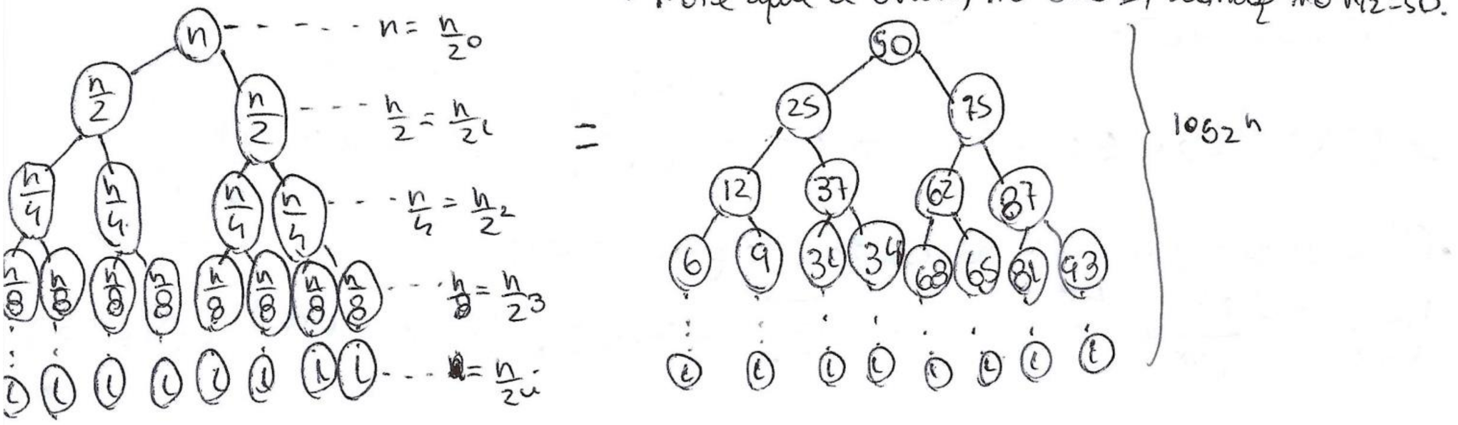
Como nós temos uma função recursiva que, a cada iteração, $X > T(\lfloor n/2 \rfloor)$ ou $X < T(\lfloor n/2 \rfloor)$, isto é, duas chamadas recursivas, temos que $T(n) = 2T(\lfloor n/2 \rfloor)$. Entretanto, há também custos constantes durante a chamada. Portanto, $T(n) = 2T(\lfloor n/2 \rfloor) + O(1)$.

∴ Considerando os casos I e II, temos que:

$$T(n) = \begin{cases} 1 & \text{se } n = 100 \quad (\text{caso I}) \\ 2T(\lfloor n/2 \rfloor) + O(1) & \text{se } n < 100 \quad (\text{caso II}) \end{cases}$$

• Árvore de recursão:

• Note que a árvore, no caso II, começa no $n/2 = 50$.



• Observe que a sequência acima só irá terminar quando chegar em 1.

Portanto, $\frac{n}{2^i} = 1 \Rightarrow i = \log_2 n$ (altura da árvore).

Como nós teremos uma tentativa para cada nível da árvore, que por sua vez tem a altura de $\log_2 n$, temos que, no pior caso:

nº de tentativas = $\log_2 n$ => nº de tentativas = $\log_2(100)$ ≅ 6. ■

4) B) Como temos apenas 2 celulares para o teste, usaremos o 1º deles para descobrir qual é o limite superior para uma quantidade n de andares. Além disso, note que no item (a) começamos pelo 100º andar, uma vez que a quantidade de dispositivos era ilimitada. Entretanto, pra esse caso específico começaremos na parte inferior do prédio.

Pior cenário possível: realizar tentativas nos 100 andares.

Assim como foi citado acima, iremos utilizar o primeiro celular para descobrir, aproximadamente, qual é o limite de andares que ele suporta. Sendo assim, se dividirmos esse prédio em 10 testes de 1 andar, começando do 10º e indo até o 100º, temos 10 tentativas base (B).

T (n/10) => (10º, 20º, 30º, 40º, 50º, 60º, 70º, 80º, 90º, 100º).

Logo, se B = 80, então sabemos que o andar X está entre 70 e 80.

Entretanto, pra saber exatamente em qual andar X o celular quebra, é necessário dividir esse andar B em 10 (I).

T(n/10) = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

Logo, se B = 80, sabemos que o andar X está entre 70 e 80 => testaremos o 2º celular do andar 71º até o 80º andar.

Assim, seja f (n) uma função que representa a integridade do celular no nº andar, B uma constante que diz respeito em qual das 10 tentativas base ele quebrou e I qual andar de B, iremos utilizar o 2º celular para jogar dos andares $((B-10)_0)$ até $((B-10)_{10})$. Ou seja, no pior cenário possível iremos realizar 10 tentativas base + 10 tentativas de I cada andar = 20 tentativas (100º andar).

Portanto, basicamente o B descobre a dezena e o I a unidade em que X está, tal que $X = (B-10)_i$;

Então por exemplo, 79º andar => B = 80-10 => B = 70 e I = 9.

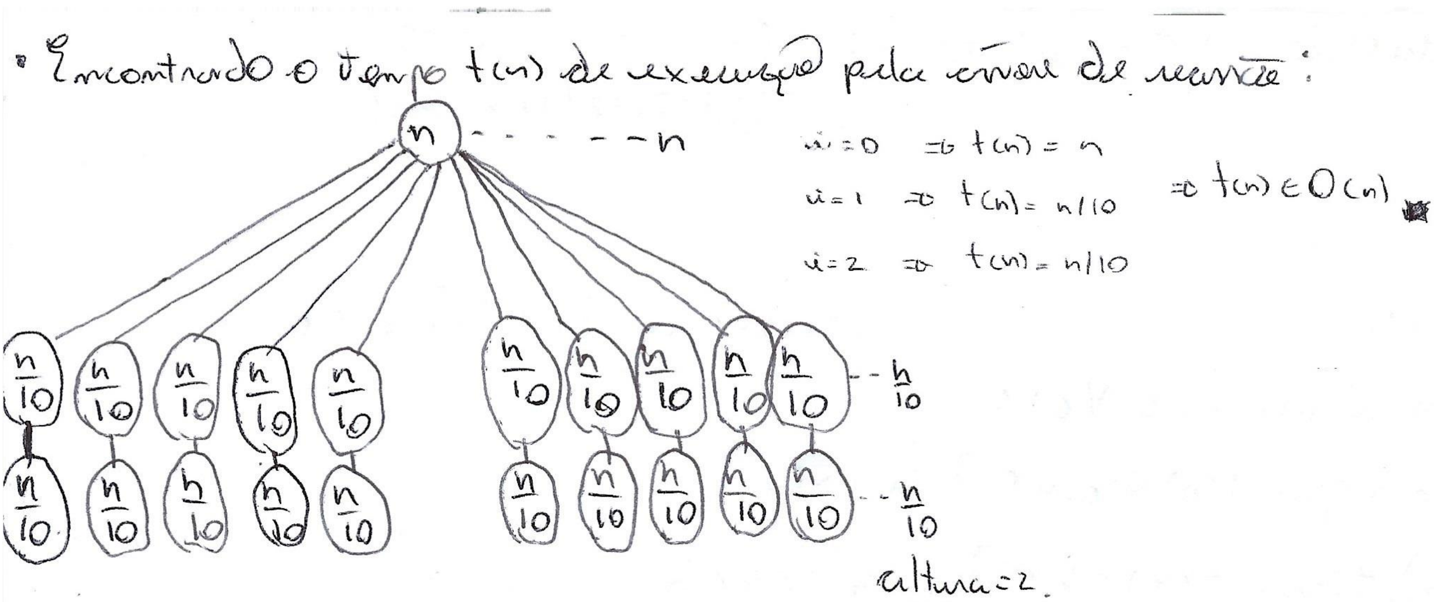
Exemplo: suponha que o celular quebre no 71º andar:

- B= 10 => F (10) = falso (não quebrou);
- B= 20 => F (20) = falso (não quebrou);
- B= 30 => F (30) = falso (não quebrou);
- B= 40 => F (40) = falso (não quebrou);
- B= 50 => F (50) = falso (não quebrou);
- B= 60 => F (60) = falso (não quebrou);
- B= 70 => F (70) = falso (não quebrou);
- B= 80 => F (80) = true (quebrou o 1º celular);
- I = 1 => F $((B-10)_i)$ = F (71) = true (quebrou 2º celular);
- Total = 8 + 1 = 9 tentativas.

Nº tentativas = (B/10) + I;

Exemplo: suponha que A) X = 12º, B) X = 15º, C) X = 82, de modo que X é o andar exato em que o celular quebra:

- A) Nº tentativas = (B/10) + I => Nº = ((20/10)) + 2 = 4;
    - B= 10 => F (10) = false (não quebrou);
    - B= 20 => F (20) = true (celular quebrou);
    - I =1 => F (11) = false (não quebrou);
    - I =2 => F (12) = true (celular quebrou);
- B) Nº tentativas = (B/10) + I => Nº = (20/10) + 5 = 7;
- C) Nº tentativas = (B/10) + I => Nº = (90/10) + 2 = 11;



Assim como foi citado acima, no pior cenário possível iremos realizar 10 tentativas base + 10 tentativas de I cada andar = 20 tentativas (100º andar). ■