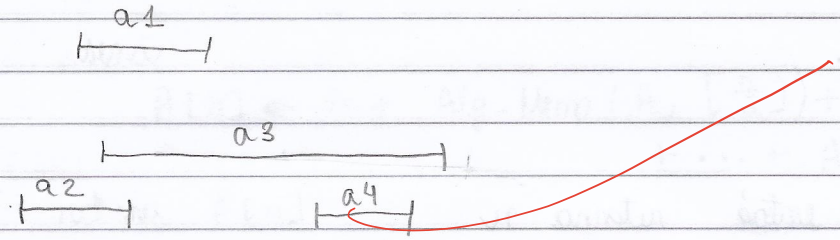
Fernanda Costa de Sousa - 485404

3,0

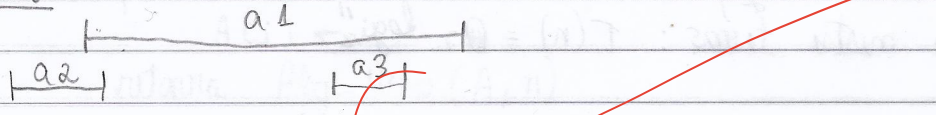
Questão 2.

* Ideia 1:



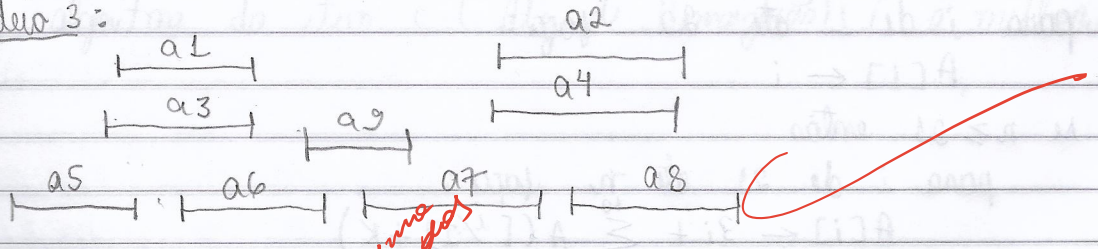
- Escolhendo sempre a atividade de maior duração, a ideia 1 escolheria a atividade a3 primeiro e não poderia escolher nenhuma outra. Sendo que uma solução ótima seria escolher as atividades a1 e a4 por exemplo.

* Ideia 2:



- Escolhendo sempre a atividade que termina por último, a ideia 2 escolheria a atividade a1, e depois não poderia escolher nenhuma outra, sendo que uma solução ótima seria escolher as atividades a2 e a3, por exemplo.

* Ideia 3:



- A atividade a9 é a que tem menos conflitos, mas se a selecionarmos, só poderemos selecionar mais 2 atividades, totalizando 3 atividades. No entanto, uma solução ótima seria a5, a6, a7 e a8, com 4 atividades.

2,1

Questão 3. $P(n) = 3n + \sum_{k=0}^{10} P\left(\left\lfloor \frac{n}{2} \right\rfloor + k\right)$

- $P(n) = n$ p/ $n = 0, 1, \dots, 20$
- $P(n) = 3n + \sum_{k=0}^{10} P(\lfloor n/2 \rfloor + k)$ p/ $n \geq 21$

a) Algoritmo recursivo:

0,7

```
Alg_Rec (n)
   se n ≤ 20 então retorna n
   senão se n ≥ 21 então
      retorna 3n + Σ_{K=0}^{10} Alg_Rec((n/2) + K)
```

?

Lembre (fez isso no algoritmo 2)

• Complexidade:

$T(n) = O(1) + T\left(\left(\frac{n}{2}\right)+0\right) + T\left(\left(\frac{n}{2}\right)+1\right) + \dots + T\left(\left(\frac{n}{2}\right)+10\right)$

$T(n) = O(1) + 11\,T\left(\frac{n}{2}\right)$

pelo teorema mestre temos: $T(n) = \Theta\, n^{\log_2 11}$

b) Algoritmo Prog. Dinâmica:

0,8

```
Alg_Prog_Din (n)
   A ← vetor [n]
   se n ≥ 0 e n ≤ 20 então retorna n
   para i de 1 até 20 faça:
      A[i] ← i
   se n ≥ 21 então
      para i de 21 até n faça:
         A[i] ← 3i + Σ_{K=0}^{10} A([i/2] + K)
   retorna A[n]
```

?

• Complexidade: $T(n) = \Theta(n)$

c) Algoritmo de memoização (recursivo usando memória)

Alg_Memo (A, n)
  se A[n] ≠ -1 (for diferente) faça:
    retorna A[n]

  senão
    A[n] ← 3n + Alg_Memo (A, $[\frac{n}{2}]$) + Alg_Memo (A, $[\frac{n}{2}]+1$)
      + ... + Alg_Memo (A, $[\frac{n}{2}]+10$)

  retorna A[n]

Algoritmo_Memoização (n)
  A ← vetor [n]
  i ← n
  A[n] ← -1
  para i de 1 até 20 faça
    A[i] ← i
  retorna Alg_Memo (A, n)

Precisa fazer isso para todos os possíveis vetores.

Complexidade: $T(n) = \Theta(\log(n))$ ✗

* Comparando os tempos dos três algoritmos podemos concluir que o algoritmo do item c (Alg. de Memoização) é o melhor. ✗

3,8

Questão 1

a) O algoritmo Y pega um array A[p..r] e vai buscar pelo menor elemento. Ele divide o array no meio e chama recursivo para a primeira e segunda metade do array, m1 e m2, enquanto elas possuem mais de um elemento.
O algoritmo Y retorna o próprio elemento no caso base em que o array só possui um elemento, caso contrário, retorna o menor elemento das duas metades m1 e m2.

O algoritmo X percorre o array A[1..n] e para cada posição i ele chama o algoritmo Y para saber quem é o menor elemento do array A[i..n] e o salva na variável j. Em seguida troca os valores da posição i com a posição j.
Podemos ver que os 2 algoritmos trabalhando em conjunto vão ordenar o vetor A ao final.

b) $T(n) = t(\frac{n}{2}) + t(\frac{n}{2}) = 2 \cdot t(\frac{n}{2})$. Olhando pelo teorema mestre temos complexidade linear $O(n)$.

c) O algoritmo X percorre o vetor A em tempo linear n e chama para cada iteração, o algoritmo Y que tem tempo de pior caso $O(N)$. Logo, o tempo final do algoritmo X é $O(n) \cdot O(n) = O(n^2)$.

d) Temos as variáveis i e j que estão mudando.
i vai iterar o vetor e j sempre vai guardar o menor elemento de A[i..n]. Na linha 3, estamos trocando a posição i e j de lugar, isso faz com que o menor elemento de A[i..n] seja sempre colocado na posição i de A[0..i]. Isso deixa um array ordenado em ordem crescente da posição A[0..i-1] (antes da linha 3) e A[0..i] após a linha 3.

Indução em que?
base?

/ /

HI: na iteração i, o vetor A está ordenado de 1 até i.

Seja a iteração $i \leftarrow i+1$

$j \leftarrow Y(A, i, n)$. j acha o menor valor em A considerando de $\{i, \ldots, n\}$. trocamos $A[i] \leftrightarrow A[j]$. Agora A está ordenado de 1 até $i+1$.

tendo $i = n$

$j \leftarrow Y(A, n, n)$ retorna n

troca $A[n] \leftrightarrow A[n]$.