1ª Atividade Avaliativa em Laboratório
Recuperação
2 de maio de 2019

- Esta atividade possui duração de 100 minutos e é individual e sem consulta.
- O estudante não precisa entregar esta folha nem as de rascunho, apenas o caderno de soluções contendo em todas as páginas o nome do estudante e o número da página.
- Cada solução apresentada deve indicar explicitamente o número da atividade à qual se refere, sem a necessidade de copiar o enunciado da atividade, e sem a necessidade de a ordem das soluções seguir a ordem das atividades.
- Junto com a identificação do número de uma atividade deve constar o peso escolhido para aquela atividade, dentro do intervalo de pesos pré-definidos. Caso os pesos de todas as atividades não somem 1, ou caso algum peso viole seu intervalo correspondente, serão atribuídos a todas as atividades os seus respectivos pesos-padrão.
- Só será considerado para fins de avaliação o que estiver escrito no caderno de soluções a tinta azul ou preta, a menos que esteja com ~~tachado duplo~~ ou ilegível.
- Poderão ser anexados arquivos às soluções das atividades, os quais devem ser entregues junto com o caderno de soluções e devidamente referenciados nele. O nome de cada arquivo deve seguir o formato `Nome-Numero.ext`, sendo `Nome` o nome do estudante completo, sem espaços nem diacríticos, `Numero` o número da atividade, e `ext` a extensão do arquivo (e.g. `LuizInacioLulaDaSilva-2.c`).
- O estudante que quiser ir ao banheiro ou entregar suas soluções, deverá primeiro levantar o braço e aguardar o professor ou encarregado.

Atividade 1 (intervalo: 0,2–0,25; padrão: 0,22)**.** Escreva uma função em C ANSI de protótipo

```
int bin_search(int x, int v[], int n);
```

que, ao receber um inteiro x e um *array* $\boldsymbol{v}[]$ de n inteiros garantidamente já ordenados em ordem não-crescente (isto é, $\boldsymbol{v}[n-1] \geq \boldsymbol{v}[n-2] \geq \cdots \geq \boldsymbol{v}[1] \geq \boldsymbol{v}[0]$), busca eficientemente pelo elemento x no *array* $\boldsymbol{v}[]$ e devolve:

- um inteiro i tal que $\boldsymbol{v}[i] = x$, caso haja ao menos uma ocorrência de x em $\boldsymbol{v}[]$;
- o inteiro -1, caso não haja ocorrência alguma de x em $\boldsymbol{v}[]$.

Resolução do Professor. Implementação iterativa:

```
int bin_search(int x, int v[], int n) {
  int a = 0, b = n - 1, m;
  while (a <= b) {
    m = (a + b) / 2;
    if (v[m] == x) return m;
    if (v[m] < x) b = m - 1;
    else a = m + 1;
  }
  return -1;
}
```

Implementação recursiva:

```
int bin_search(int x, int v[], int n) {
  int m = (n - 1) / 2, aux;
  if (n <= 0) return -1;
  if (v[m] == x) return m;
  if (v[m] < x) return bin_search(x, v, m);
  aux = bin_search(x, v + m + 1, n - m + 1);
  return aux + (aux != -1) * (m + 1);
}
```

Atividade 2 (intervalo: 0,2–0,3; padrão: 0,23)**.** Considere a seguinte função implementada em C ANSI, a qual, ao receber um número natural n, devolve 1 se n é primo ou 0 caso contrário.

```c
int is_prime(unsigned long long n) {
  int i;
  if (n <= 1) return 0;
  for (i = 2; i < n; i++)
    if (n % i == 0) return 0;
  return 1;
}
```

Expresse usando notação assintótica a complexidade de tempo da função `is_prime()` no melhor caso e no pior caso. Podemos dizer que o algoritmo que ela implementa é um algoritmo linear no tamanho da entrada? Justifique, apresentando as complexidades em função do tamanho da entrada.

Resolução do Professor. No melhor caso, quando $n \leq 1$ ou n é par, a função realiza apenas $O(1)$ operações, devolvendo a resposta já na primeira linha ou na primeira iteração do laço. No pior caso, quando n é primo, a função realiza $O(n)$ operações, pois o laço é executado para todo i de 2 até $n-1$, e a função devolve a resposta apenas na última linha. Não podemos dizer que o algoritmo implementado é linear, uma vez que n é *a* entrada, cujo tamanho é $m := \lfloor \lg n \rfloor + 1$. Logo, em função do tamanho da entrada m, as complexidades de tempo no melhor e no pior caso são, respectivamente, $O(1)$ e $O(2^m)$.

Atividade 3 (intervalo: 0,15–0,25; padrão: 0,15)**.** Escreva um programa em C ANSI que lê da entrada padrão um número natural n, aloca dinamicamente um vetor com n posições do tipo *ponteiro para inteiro* (`int *`), libera então todo o espaço de memória alocado, e termina.

Resolução do Professor.

```c
#include<stdio.h>
#include<stdlib.h>

int main(void) {
  int n, **v;
  scanf("%d", &n);
  v = (int **)malloc(n * sizeof(int *));
  free(v);
  return 0;
}
```

Atividade 4 (intervalo: 0,2–0,3; padrão: 0,2)**.** Ordene as seguintes funções de modo que para cada duas funções f e g consecutivas na sua ordem valha que $f(n) = o(g(n))$. Não é necessário justificar sua resposta.

$$n\log n \qquad \log n \qquad 1 \qquad 3^n \qquad n^2 \qquad 2^{n^2} \qquad n \qquad \sqrt{n} \qquad n^3 \qquad 2^n \qquad \frac{1}{n}$$

Resolução do Professor.

$$\frac{1}{n} \qquad 1 \qquad \log n \qquad \sqrt{n} \qquad n \qquad n\log n \qquad n^2 \qquad n^3 \qquad 2^n \qquad 3^n \qquad 2^{n^2}$$

Atividade 5 (intervalo: 0–0,4; padrão: 0,2)**.** Mostre que se $f : \mathbb{N} \to \mathbb{R}_+ \cup \{0\}$ é uma função de complexidade tal que $f(n) = O(\log n)$, então existe uma constante real positiva c tal que para todo $n \geq 2$ vale que $f(n) \leq c \lg n$.

Resolução do Professor. Temos que existem um natural n_0 e uma constante real $r > 0$ tais que $f(n) \leq r \lg n$ para todo $n \geq n_0$. Queremos mostrar que existe uma constante real positiva c tal que $f(n) \leq c \log n$ para todo $n \geq 2$. Ora, supondo sem perda de generalidade que $n_0 \geq 2$, e sendo F a constante definida por $F := 1 + \max_{0 \leq i \leq n_0} f(i)$, sabemos que, para todo natural $n \geq 2$,

$$f(n) \leq \begin{cases} F \leq F \lg n, & \text{se } n \leq n_0, \\ r \lg n, & \text{caso contrário.} \end{cases}$$

Portanto, a alegação se verifica bastando tomar $c := \max\{F, r\}$.