DEEC
DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA E DE COMPUTADORES
TÉCNICO LISBOA

AED - 2018-2019 - 2º Semestre
Algoritmos e Estruturas de Dados

Laboratório de Avaliação Individual

Semana de 25 de Março de 2019 Duração: 1h30mn

Este laboratório consiste numa avaliação prática **individual** e será realizado obrigatoriamente nos computadores da sala, usando a área de utilizador `aed`.

A avaliação envolve a resolução de **três problemas** cuja descrição formal se segue. Nalguns casos o problema é apenas parcialmente apresentado aqui, sendo a descrição completa disponibilizada no momento da avaliação. O objectivo da divulgação do enunciado (completo ou parcial) dos problemas é tornar a avaliação mais justa e independente da altura em que cada aluno é avaliado.

---

1. Considere a função `check_property()` abaixo, que determina a quantos "níveis" um vector `vec`, de tamanho N, satisfaz uma certa propriedade. Neste caso a propriedade testada é **se os números de cada (sub-)tabela sao iguais entre si** e os "níveis" a que o teste é efectuado correspondem a sub-divisões sucessivas da tabela em duas.

Se a propriedade é válida na tabela original, então o "nível" é 1, mas se dividindo a tabela em duas metades a mesma propriedade é também válida em ambas as metades, então o nível já é 2; por outro lado se a propriedade não é válida na tabela original mas é válida nas suas duas metades, então o "nível" é de novo 1, etc.

A função é inicialmente chamada da seguinte forma: `check_property(vec, 0, N-1)`

```
int check_property(int *vec, int iL, int iR) {
  int k, res, resl, resr;
  int iM = (iR - iL) / 2;

  if (iM == 0) {
    if (vec[iL] != vec[iR]) return(0);
    else return(1);
  }
  for (res = 1, k = 0; k <= (iR - iL); k++) {
    if (vec[iL + k] != vec[iL + k + 1]) {
      res = 0; break;
    }
  }
  resl = check_property(vec, iL, iL + iM);
  resr = check_property(vec, iL + iM+1, iR);
  return(res + min(resl,resr));
}
```

Nota: a propriedade indicada é apenas ilustrativa. Nos enunciados da avaliação será dado código que verifica propriedades diferentes.

Escreva no espaço abaixo a recorrência que descreve a complexidade C_N da execução do código em função de N (assuma que N é uma potencia de 2). `Justifique` a fórmula que escreveu.

2. O código incompleto mostrado abaixo (disponível em `LabAval/p2/p2.c`) recebe como argumento o nome de um ficheiro. O conteúdo deste começa por dois inteiros que indicam as dimensões de uma matriz, seguido dos valores inteiros que constituem essa matriz.

O programa deverá ler o ficheiro de entrada e alocar a memória necessária para a matriz, processar a matriz conforme especificado e imprimir o resultado desse processamento, após o que deverá libertar completamente a memória alocada antes de terminar.

Complete o código, incluindo as instruções necessárias para alocação e libertação de memória, bem como o **código da função solicitada no seu enunciado** sabendo que o seu protótipo deve ser:

```
void funcao(Matriz *, int *, int *);
```

Por exemplo, a funcionalidade pedida poderia ser a de calcular a soma de todos os elementos da matriz e a soma dos módulos de todos os elementos da matriz. As variáveis `out1` e `out2` deverão receber os valores pedidos (veja o código disponibilizado).

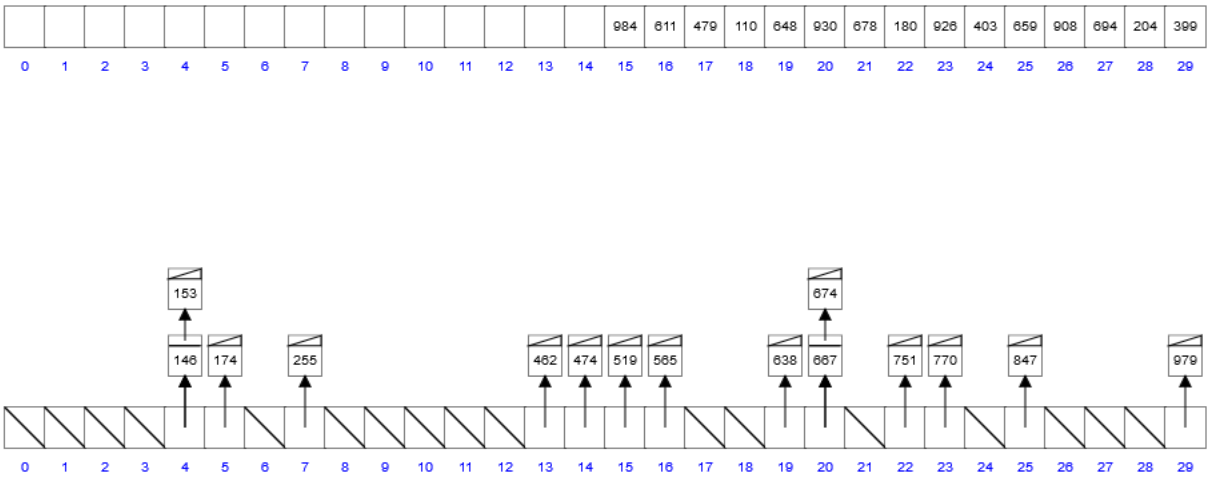
Nota: a funcionalidade acima é apenas ilustrativa. Na avaliação será pedida uma funcionalidade ligeiramente diferente, distinta para cada horário.

```
typedef struct _matriz {
    int linhas;
    int colunas;
    int ** matriz;
} Matriz;

void funcao(Matriz *M, int *a1, int *a2) {
/* to write during the lab */
}

int main(int argc, char *argv[]) {

  int i, j;
  int out1, out2;
  Matriz MAT;
  FILE * fpin;

  if (argc < 2) {
    fprintf(stderr, "Usage:  %s <file>...\n", argv[0]);
    exit(1);
  }

  /* open file and read dimensions */
  fpin = fopen(argv[1], "r");
  fscanf(fpin, "%d %d", &MAT.linhas, &MAT.colunas);

  /* allocate memory, set structure, read in array */
  MAT.matriz = (int **) malloc(/* ...  */);
  ...

  for (i = 0; i < MAT.linhas; i++)
    for (j = 0; j < MAT.colunas; j++)
      fscanf(fpin, "%d", &MAT.matriz[i][j]);

  funcao(&MAT, &out1, &out2);
  printf("%d %d \n", out1, out2);

  /* free memory */

  exit(0);
}
```

3. O algoritmo de ordenação designado por *Bucket sort* depende de separar os dados recebidos em 'baldes'. Cada balde tem uma lista ligada em que é feita inserção ordenada, como ilustrado na figura abaixo

Por exemplo, para ordenar uma tabela de N inteiros positivos, existe uma tabela de M listas ligadas (baldes). A ordenação é feita em dois passos:

- Percorrer a tabela de inteiros: para cada valor obter o índice do seu balde (por exemplo $i = M * val/(maxval + 1)$), e inserir ordenadamente na lista respectiva.
- Percorrer a tabela de listas, lendo cada lista ordenada. A sequência de valores obtida é a nova ordenação da tabela de inteiros, e pode ser escrita directamente para esta.



(figura cortesia de `https://www.cs.usfca.edu/~galles/visualization/BucketSort.html`)

Neste exercício solicita-se a implementação de *Bucket sort* e a análise de algumas das suas características em várias circunstâncias.

Na pasta `LabAval/p3/` é fornecido código de manipulação de listas simplesmente ligadas, incluindo uma função de inserção ordenada, bem como o esqueleto base da função principal que lê dados de teste.

3.1. Implemente o *Bucket sort* para inteiros positivos lidos de um ficheiro de texto.

Complete o código fornecido em `p3.c` de modo a que os dados lidos sejam ordenados por *Bucket sort* em ordem crescente e escritos para o terminal.

3.2. Escreva no espaço abaixo a sua análise da complexidade de execução do algoritmo se os dados forem ... / o número de baldes utilizados for ...

3.3. Desenvolva uma variante `p33.c` do programa original `p3.c` que seja capaz de lidar eficazmente com dados do seguinte tipo ... / obedecendo às seguintes condições ...

Na avaliação, o enunciado especificará completamente as condições elididas acima