Lista 13 - Francisco Braz

1-)
a-)
OBS: a função ascii() seria uma função que converte inteiro em char com base na tabela ascii, então dependeria da linguagem

```
{-Entrada: inteiro n, representando o tamanho da senha e vetor A de tamanho n, que será o vetor que guardará a senha.
Saída: Vetor senha gerada-}
início
    função gerarSenha(A, n): vazio
    início
        temNum := false
        temEsp := false
        temMaiusc := false
        temMinusc := false

        para i = 1 até n faça
        início
            A[i] = caracter(temNum, temEsp, temMaiusc, temMinusc)
        fim

        checarSenha(A, n, temNum, temEsp, temMaiusc, temMinusc)
    fim

    função caracter(temNum, temEsp, temMaiusc, temMinusc): char
    início
        x := random() % 95
        se x <= 14 então temEsp = true
        senão se x >= 15 && x <= 24 então temNum = true
        senão se x >= 25 && x <= 31 então temEsp = true
        senão se x >= 32 && x <= 57 então temMaiusc = true
        senão se x >= 58 && x <= 63 então temEsp = true
        senão se x >= 64 && x <= 89 então temMinusc = true
        senão temEsp = true
```

```
        x = x + 33
        y := ascii(x)
        return y
    fim

    função checar(A, n, temNum, temEsp, temMaiusc, temMinusc):vazio
    início
       enquanto
         temNum = false ||
         temEsp = false ||
         temMaiusc = false ||
         temMinusc = false
       faça gerarSenha(A, n)

        para i = 1 até n faça imprimia A[i]
        return;
    fim
fim
```

b-) Implementação no arquivo senhaForte.c

3-)

a-)

Fonte:https://www.cantorsparadise.com/calculating-the-value-of-pi-using-random-numbers-a-monte-carlo-simulation-d4b80dc12bdf

```
{-Entrada: inteiro n, representando a quantidade de pontos que iremos estimar para o cálculo de pi.
Saída: Valor estimado de pi-}
início
    funçãocalcularPi(n): double
    início
        qtd := 0.0
        para i = 1 até n faça
        início
            x := random()
```

```
            y := random()
            se (x*x) + (y*y) <= 1 então qtd = qtd + 1
        fim
        pi := 4 * (qtd/n)
        retorna pi
    fim
fim
```

b-) Implementação no arquivo calcularPi.c

4-)

a-)

Ideia da modificação: Meu algoritmo original sempre utiliza o valor do elemento na posição '*esq'* como pivô. Como, nessa versão iremos escolher um valor aleatório, eu escolho o elemento, guardando o index do elemento na variável *'index'*. Em seguida eu troco o elemento na posição *'index'* com o elemento na posição *'esq'*. O código restante é o mesmo do algoritmo "original". OBS: estou supondo que a função *random()* gera um valor entre 0 e 1.

```
função partição(X, esq, dir)
início
    index := esq + (random() % (dir-esq))

    aux := X[esq]
    X[esq] = X[index]
    x[index] = aux

    pivo := X[esq]
    L := esq
    R := dir
    enquanto L < R faça
    início
        enquanto (X[L] <= pivo) e (L <= dir) faça L := L+1
        enquanto (X[R] > pivo)  faça R := R-1
        se (L < R) então Troca(X[L], X[R])
```

```
    fim
     pos := R
     X[esq] = X[pos]
     X[pos] = pivo
     retorne pos
fim

função quickSort(X, esq, dir)
início
  se (esq < dir) então
      pos := Particao(X, esq, dir)
      quickSort(X, esq, pos-1)
      quickSort(X, pos+1, dir)
fim
```

b-) Implementação do arquivo quicksortPivoAleatorio.c