**ĐẠI HỌC HUẾ**



# KHOA KỸ THUẬT VÀ CÔNG NGHỆ

🙠🙟🕮🙝🙢

****

**BÁO CÁO**

**ĐỒ ÁN**

**NĂM HỌC 2021-2022**

**Giảng viên hướng dẫn: Hồ Quốc Dũng**

**Lớp: KHDL & TTNT**

|  |
| --- |
| Số phách  *(Do hội đồng chấm thi ghi)* |

**Thừa Thiên Huế, ngày 1 tháng 6 năm 2021**

**ĐẠI HỌC HUẾ**

# KHOA KỸ THUẬT VÀ CÔNG NGHỆ



🙠🙟🕮🙝🙢

****

(MẪU BÌA PHỤ)

**BÁO CÁO**

**ĐỒ ÁN**

**NĂM HỌC 2021-2022**

**Giảng viên hướng dẫn: Hồ Quốc Dũng**

**Lớp: KHDL & TTNT**

**Môn học: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

**Họ và tên: Ngô Nhật Khánh**

**Mã sinh viên: 20E1020011**

*(ký tên và ghi rõ họ tên)*

|  |
| --- |
| Số phách  *(Do hội đồng chấm thi ghi)* |

**Thừa Thiên Huế, ngày 11 tháng 6 năm 2021**

Mục lục

[KHOA KỸ THUẬT VÀ CÔNG NGHỆ 1](#_Toc77446912)

[KHOA KỸ THUẬT VÀ CÔNG NGHỆ 2](#_Toc77446913)

[**1. Tháp Hà Nội:** 4](#_Toc77446914)

[a. Python: 4](#_Toc77446915)

[b. R: 4](#_Toc77446916)

[**2. Ước số chung lớn nhất:** 5](#_Toc77446917)

[a. Python: 5](#_Toc77446918)

[b. R: 5](#_Toc77446919)

[**3. Tính giai thừa của 1 số:** 5](#_Toc77446920)

[a. Python: 5](#_Toc77446921)

[b. R: 6](#_Toc77446922)

[**4. Bài toán mã đi tuần:** 6](#_Toc77446923)

[a. Python: 6](#_Toc77446924)

[b. R: 7](#_Toc77446925)

[**5. Bài toán 8 quân hậu:** 8](#_Toc77446926)

[a. Python: 8](#_Toc77446927)

[b. R: 9](#_Toc77446928)

[**6. Cài đặt danh sách liên kết đơn:** 10](#_Toc77446929)

[a. Python: 10](#_Toc77446930)

[b. R: 11](#_Toc77446931)

[**7. Cài đặt danh sách liên kết kép:** 11](#_Toc77446932)

[a. Python: 11](#_Toc77446933)

[b. R: 12](#_Toc77446934)

[**8. Cài đặt ngăn xếp Stack:** 12](#_Toc77446935)

[a. Python: 12](#_Toc77446936)

[b. R: 13](#_Toc77446937)

[**9. Cài đặt hàng đợi Queue:** 13](#_Toc77446938)

[a. Python: 13](#_Toc77446939)

[b. R: 14](#_Toc77446940)

[**10. Cài đặt cây - duyệt cây theo thứ tự trước:** 14](#_Toc77446941)

[a. Python: 14](#_Toc77446942)

[b. R: 15](#_Toc77446943)

[11. Cài đặt cây - duyệt cây theo thứ tự sau: 15](#_Toc77446944)

[a. Python: 15](#_Toc77446945)

[b. R: 16](#_Toc77446946)

[**12. Cài đặt đồ thị vô hướng:** 16](#_Toc77446947)

[a. Python: 16](#_Toc77446948)

[b. R: 17](#_Toc77446949)

[**13. Cài đặt đồ thị có hướng:** 17](#_Toc77446950)

[a. Python: 17](#_Toc77446951)

[b. R: 17](#_Toc77446952)

[**14. Cài đặt thuật toán sắp xếp chọn:** 17](#_Toc77446953)

[a. Python: 17](#_Toc77446954)

[b. R: 17](#_Toc77446955)

[**15. Cài đặt thuật toán sắp xếp chèn:** 18](#_Toc77446956)

[a. Python: 18](#_Toc77446957)

[b. R: 18](#_Toc77446958)

[**16. Cài đặt thuật toán sắp xếp nổi bọt:** 19](#_Toc77446959)

[a. Python: 19](#_Toc77446960)

[b. R: 19](#_Toc77446961)

[**17. Cài đặt thuật toán sắp xếp nhanh - quick sort:** 19](#_Toc77446962)

[a. Python: 19](#_Toc77446963)

[b. R: 20](#_Toc77446964)

[**18. Cài đặt thuật toán heap sort:** 20](#_Toc77446965)

[a. Python: 20](#_Toc77446966)

[b. R: 21](#_Toc77446967)

[**19. Cài đặt thuật toán sắp xếp trộn – merge sort:** 22](#_Toc77446968)

[a. Python: 22](#_Toc77446969)

[b. R: 23](#_Toc77446970)

# 1. Tháp Hà Nội:

## a. Python:

def TowerOfHanoi(n, from\_rod, to\_rod, aux\_rod):

if n == 1:

print("Move disk 1 from rod", from\_rod, "to rod", to\_rod)

return

TowerOfHanoi(n - 1, from\_rod, aux\_rod, to\_rod)

print("Move disk", n, "from rod", from\_rod, "to rod", to\_rod)

TowerOfHanoi(n - 1, aux\_rod, to\_rod, from\_rod)

n = int(input('Nhap so dia: '))

TowerOfHanoi(n, 'A', 'C', 'B')

## b. R:

tower\_of\_hanoi <- function(n = 7) {

if (!interactive()) return()

tower <- list(1:n, NULL, NULL)

color <- rainbow(n)

par(mfrow = c(1, 3), mar = rep(0, 4), ann = FALSE)

bgcolor <- par("bg")

if (bgcolor == "transparent") bgcolor <- "white"

draw.hanoi <- function() {

for (i in 1:3) {

plot(c(-n, n), c(0, n + 2), type = "n", xlab = "",

ylab = "", axes = FALSE)

rect(-n, 0, n, n + 2, border = bgcolor, col = bgcolor)

if (length(tower[[i]]) > 0) {

barplot(rev(tower[[i]]), add = TRUE, horiz = TRUE,

col = color[rev(tower[[i]])])

barplot(-rev(tower[[i]]), add = TRUE, horiz = TRUE,

col = color[rev(tower[[i]])])

}

}

}

move.hanoi <- function(k, from, to, via) {

if (k > 1) {

move.hanoi(k - 1, from, via, to)

move.hanoi(1, from, to, via)

move.hanoi(k - 1, via, to, from)

}

else {

cat("Move ", tower[[from]][1], " from ", LETTERS[from],

" to ", LETTERS[to], "\n")

tower[[to]] <<- c(tower[[from]][1], tower[[to]])

tower[[from]] <<- tower[[from]][-1]

draw.hanoi()

Sys.sleep(0.5)

}

}

draw.hanoi()

move.hanoi(n, 1, 2, 3)

}

tower\_of\_hanoi(8)

# 2. Ước số chung lớn nhất:

## a. Python:

def uscln(a, b):

if a < 0 or b < 0:

return None

if b == 0:

return a

return uscln(b, a % b)

a = int(input("Nhập số nguyên dương a = "))

b = int(input("Nhập số nguyên dương b = "))

print("Ước số chung lớn nhất của", a, "và", b, "là:", uscln(a, b))

## b. R:

uscln <- function(a, b){

if (a < 0 | b < 0){

return('None exists')

}

if (b == 0){

return(a)

}

else {

return(uscln(b, a %% b))

}

}

uscln(3, -2)

# 3. Tính giai thừa của 1 số:

## a. Python:

def tinhgiaithua(n):

giai\_thua = 1;

if n < 0:

return ('Không tồn tại')

elif n == 0 or n == 1:

return giai\_thua

else:

for i in range(2, n + 1):

giai\_thua = giai\_thua \* i

return giai\_thua

n = int(input("Nhập số nguyên dương n = "))

print("Giai thừa của", n, "là", tinhgiaithua(n))

## b. R:

giaithua <- function(n){

giai\_thua = 1

if (n < 0){

return('None exists')

}

else if (n == 0 | n == 1){

return(giai\_thua)

}

else {

return(n \* giaithua(n - 1))

}

}

giaithua(3)

# 4. Bài toán mã đi tuần:

## a. Python:

n = int(input('Ti le n = '))

def isSafe(x, y, board):

if (x >= 0 and y >= 0 and x < n and y < n and board[x][y] == -1):

return True

return False

def printSolution(n, board):

for i in range(n):

for j in range(n):

print(board[i][j], end=' ')

print()

def solveKT(n):

board = [[-1 for i in range(n)] for i in range(n)]

move\_x = [2, 1, -1, -2, -2, -1, 1, 2]

move\_y = [1, 2, 2, 1, -1, -2, -2, -1]

board[0][0] = 0

pos = 1

if (not solveKTUtil(n, board, 0, 0, move\_x, move\_y, pos)):

print("Solution does not exist")

else:

printSolution(n, board)

def solveKTUtil(n, board, curr\_x, curr\_y, move\_x, move\_y, pos):

if (pos == n \*\* 2):

return True

for i in range(8):

new\_x = curr\_x + move\_x[i]

new\_y = curr\_y + move\_y[i]

if (isSafe(new\_x, new\_y, board)):

board[new\_x][new\_y] = pos

if (solveKTUtil(n, board, new\_x, new\_y, move\_x, move\_y, pos + 1)):

return True

board[new\_x][new\_y] = -1

return False

solveKT(n)

## b. R:

knight\_offsets <- matrix(c(1, 2, 2, 1, -2, 1, -1, 2, 2, -1, 1, -2, -1, -2, -2, -1),

ncol = 2, byrow = TRUE)

move\_knight <- function(this\_move, moves, visited) {

moves <- append(moves, list(this\_move))

visited[this\_move[1] + (this\_move[2] - 1)\*8] <- TRUE

if (all(visited)) {

return(moves)

}

next\_move <- cbind(knight\_offsets[,1] + this\_move[1],

knight\_offsets[,2] + this\_move[2])

on\_board <- next\_move[,1] %in% 1:8 & next\_move[,2] %in% 1:8

next\_move <- next\_move[on\_board,,drop=FALSE]

not\_yet\_visited <- !visited[next\_move]

next\_move <- next\_move[not\_yet\_visited,, drop = FALSE]

for (i in seq\_len(nrow(next\_move))) {

res <- move\_knight(next\_move[i,, drop = FALSE], moves, visited)

if (!is.null(res)) {

return(res)

}

}

NULL

}

system.time({

moves <- move\_knight(c(4, 8), moves = list(), visited = matrix(FALSE, 8, 8))

})

moves\_df <- as.data.frame(do.call(rbind, moves))

moves\_df <- set\_names(moves\_df, c('x', 'y'))

moves\_df$idx <- 1:nrow(moves\_df)

ggplot(moves\_df, aes(x, y)) +

geom\_tile(aes(fill=as.logical((x+y)%%2)), colour = 'black') +

geom\_path(alpha = 0.7, linetype = 1, size = 0.25) +

geom\_text(aes(label = idx)) +

scale\_fill\_manual(values = c('grey70', 'white')) +

theme\_void() +

theme(legend.position = 'none') +

coord\_equal() +

labs(title = "A knight's tour in #RStats")

# 5. Bài toán 8 quân hậu:

## a. Python:

import numpy as np

import copy

import seaborn as sns

import matplotlib.pyplot as plt

import string

N = 10

grid = np.zeros([N, N], dtype=int)

grid = grid.tolist()

def possible(grid, y, x):

l = len(grid)

for i in range(l):

if grid[y][i] == 1:

return False

for i in range(l):

if grid[i][x] == 1:

return False

for i in range(l):

for j in range(l):

if grid[i][j] == 1:

if abs(i - y) == abs(j - x):

return False

return True

def solve(grid):

l = len(grid)

for y in range(l):

for x in range(l):

if grid[y][x] == 0:

if possible(grid, y, x):

grid[y][x] = 1

solve(grid)

if sum(sum(a) for a in grid) == l:

return grid

grid[y][x] = 0

return grid

Solution = solve(copy.deepcopy(grid))

print(np.matrix(Solution))

def plot(grid):

l = len(grid)

Ly = list(range(1, l + 1))[::-1]

ly = [str(i) for i in Ly]

Lx = list(string.ascii\_uppercase)

lx = Lx[:l]

plt.close('all')

sns.set(font\_scale=2)

plt.figure(figsize=(10, 10))

ax = plt.gca()

ax.set\_aspect(1)

sns.heatmap(Solution, linewidths=.8, cbar=False, linecolor='blue',

cmap='Reds', center=0.4, xticklabels=lx, yticklabels=ly)

plot(grid)

## b. R:

library(tidyverse)

place\_queen <- function(queens=c()) {

if (length(queens) == 8) {

return(list(queens))

}

possible\_placements <- setdiff(1:8, queens)

diag\_offsets <- seq.int(length(queens), 1)

diags <- c(queens + diag\_offsets, queens - diag\_offsets)

diags <- diags[diags > 0 & diags < 9]

possible\_placements <- setdiff(possible\_placements, diags)

possible\_placements %>%

map(~place\_queen(c(queens, .x))) %>%

keep(~length(.x) > 0) %>%

flatten()

}

plot\_single\_8queens <- function(queens, title = NULL) {

queens\_df <- tibble(cols = queens, rows=1:8)

board\_df <- expand.grid(cols = 1:8, rows = 1:8) %>%

mutate(check = (cols + rows) %%2 == 1)

p <- ggplot(queens\_df, aes(rows, cols)) +

geom\_tile(data=board\_df, aes(fill=check), colour='black') +

geom\_text(label='\u2655', family="Arial Unicode MS", size = 8) +

theme\_void() +

coord\_equal() +

scale\_fill\_manual(values = c('TRUE'='white', 'FALSE'='grey70')) +

theme(

legend.position = 'none'

)

if (is.null(title)) {

p <- p + labs(title = paste("Queens:", deparse(as.numeric(queens))))

} else {

p <- p + labs(title = title)

}

}

solutions <- place\_queen()

v=1:8

f=function(q){L=length(q)

if(L==8){q}else{flatten(map(setdiff(v,c(q,q+L:1,q-L:1)),~f(c(q,.))))}}

s=data.frame(c=unlist(f(c())),r=v,x=rep(1:92,e=8),z=3)

b=mutate(crossing(c=v,r=v),z=(c+r)%%2)

g=geom\_tile

ggplot(s,aes(r,c,fill=z))+g(d=b)+g()+facet\_wrap(~x)

# 6. Cài đặt danh sách liên kết đơn:

## a. Python:

class Node(object):

def \_\_init\_\_(self):

self.data = None # contains the data

self.next = None # contains the reference to the next node

class LinkedList:

def \_\_init\_\_(self):

self.cur\_node = None

def add\_node(self, data):

new\_node = Node() # create a new node

new\_node.data = data

new\_node.next = self.cur\_node # link the new node to the 'previous' node.

self.cur\_node = new\_node # set the current node to the new one.

def list\_print(self):

node = self.cur\_node # cant point to ll!

while node:

print (node.data)

node = node.next

ll = LinkedList()

ll.add\_node(343)

ll.add\_node(546)

ll.add\_node(324)

ll.list\_print()

## b. R:

lst <- list()

lst[[1]] <- 1

lst[[2]] <- 2

lst[[3]] <- 3

lst[[4]] <- 4

lst

# 7. Cài đặt danh sách liên kết kép:

## a. Python:

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

self.prev = None

class DoublyLinkedList:

def \_\_init\_\_(self):

self.head = None

def push(self, new\_data):

new\_node = Node(new\_data)

new\_node.next = self.head

if self.head is not None:

self.head.prev = new\_node

self.head = new\_node

def insertAfter(self, prev\_node, new\_data):

if prev\_node is None:

print("the given previous node cannot be NULL")

return

new\_node = Node(new\_data)

new\_node.next = prev\_node.next

prev\_node.next = new\_node

new\_node.prev = prev\_node

if new\_node.next is not None:

new\_node.next.prev = new\_node

def append(self, new\_data):

new\_node = Node(new\_data)

new\_node.next = None

if self.head is None:

new\_node.prev = None

self.head = new\_node

return

last = self.head

while (last.next is not None):

last = last.next

last.next = new\_node

new\_node.prev = last

return

def printList(self, node):

print("\nTraversal in forward direction")

while (node is not None):

print(" % d" % (node.data), )

last = node

node = node.next

print("\nTraversal in reverse direction")

while (last is not None):

print(" % d" % (last.data), )

last = last.prev

llist = DoublyLinkedList()

llist.append(6)

llist.push(7)

llist.push(5)

llist.append(4)

llist.insertAfter(llist.head.next, 8)

print("Created DLL is: ", )

llist.printList(llist.head)

## b. R:

lst <- list(1, 2, 3, 4, 5)

lst <- vector("list", 10000)

lst[[1]] <- 1

lst[[10000]] <- 10000

lst

# 8. Cài đặt ngăn xếp Stack:

## a. Python:

class Stack:

def \_\_init\_\_(self):

self.stack = []

def add(self, dataval):

if dataval not in self.stack:

self.stack.append(dataval)

return True

else:

return False

def peek(self):

return self.stack[-1]

AStack = Stack()

AStack.add("Mon")

AStack.add("Tue")

AStack.peek()

print(AStack.peek())

print('-'\*10)

AStack.add("Wed")

AStack.add("Thu")

print(AStack.peek())

## b. R:

library(dequer)

s <- stack()

for (i in 1:3) push(s, i)

str(s)

pop(s)

str(s)

pop(s)

str(s)

pop(s)

str(s)

# 9. Cài đặt hàng đợi Queue:

## a. Python:

class Queue:

def \_\_init\_\_(self):

self.items = []

def isEmpty(self):

return self.items == []

def pop(self):

return self.items.pop(0)

def size(self):

return len(self.items)

def peek(self):

return self.items[ len(self.items) - 1 ]

def push(self, item):

self.items.append(item)

q = Queue()

print ("Is empty: " + str(q.isEmpty()))

q.push(1)

q.push(2)

q.push(3)

print ("List: " + str(q.items))

print ("Pop item at position 0: " + str(q.pop()))

print ("List: " + str(q.items))

print ("Peek of queue: " + str(q.peek()))

print ("Size: " + str(q.size()))

## b. R:

library(dequer)

q <- queue()

for (i in 1:3) pushback(q, i)

str(q)

pop(q)

str(q)

pop(q)

str(q)

pop(q)

str(q)

# 10. Cài đặt cây - duyệt cây theo thứ tự trước:

## a. Python:

class Node:

def \_\_init\_\_(self, data):

self.left = None

self.right = None

self.data = data

def insert(self, data):

if self.data:

if data < self.data:

if self.left is None:

self.left = Node(data)

else:

self.left.insert(data)

elif data > self.data:

if self.right is None:

self.right = Node(data)

else:

self.right.insert(data)

else:

self.data = data

def PrintTree(self):

if self.left:

self.left.PrintTree()

print( self.data),

if self.right:

self.right.PrintTree()

def inorderTraversal(self, root):

res = []

if root:

res = self.inorderTraversal(root.left)

res.append(root.data)

res = res + self.inorderTraversal(root.right)

return res

root = Node(27)

root.insert(14)

root.insert(35)

root.insert(10)

root.insert(19)

root.insert(31)

root.insert(42)

print(root.inorderTraversal(root))

## b. R:

library(data.tree)

A <- Node$new("A")

B <- A$AddChild("B")

C <- B$AddChild("C")

D <- B$AddChild("D")

E <- A$AddChild("E")

F <- E$AddChild("F")

G <- E$AddChild("G")

H <- A$AddChild("H")

I <- H$AddChild("I")

J <- H$AddChild("J")

K <- H$AddChild("K")

print(A)

A$Get('level')

# 11. Cài đặt cây - duyệt cây theo thứ tự sau:

## a. Python:

class Node:

def \_\_init\_\_(self, data):

self.left = None

self.right = None

self.data = data

def insert(self, data):

if self.data:

if data < self.data:

if self.left is None:

self.left = Node(data)

else:

self.left.insert(data)

elif data > self.data:

if self.right is None:

self.right = Node(data)

else:

self.right.insert(data)

else:

self.data = data

def PrintTree(self):

if self.left:

self.left.PrintTree()

print( self.data),

if self.right:

self.right.PrintTree()

def PreorderTraversal(self, root):

res = []

if root:

res.append(root.data)

res = res + self.PreorderTraversal(root.left)

res = res + self.PreorderTraversal(root.right)

return res

root = Node(27)

root.insert(14)

root.insert(35)

root.insert(10)

root.insert(19)

root.insert(31)

root.insert(42)

print(root.PreorderTraversal(root))

## b. R:

library(data.tree)

A <- Node$new("A")

B <- A$AddChild("B")

C <- B$AddChild("C")

D <- B$AddChild("D")

E <- A$AddChild("E")

F <- E$AddChild("F")

G <- E$AddChild("G")

H <- A$AddChild("H")

I <- H$AddChild("I")

J <- H$AddChild("J")

K <- H$AddChild("K")

print(A)

A$Get('level', traversal = "post-order")

# 12. Cài đặt đồ thị vô hướng:

## a. Python:

import pandas as pd

import networkx as nx

import matplotlib.pyplot as plt

df = pd.DataFrame({'from': ['A', 'B', 'C', 'D'], 'to': ['B', 'D', 'A', 'A']})

G = nx.from\_pandas\_edgelist(df, 'from', 'to', create\_using=nx.DiGraph())

nx.draw(G, with\_labels=True, node\_size=1500, alpha=0.3, arrows=True)

plt.title("Directed")

plt.show()

## b. R:

install.packages('igraph')

library(igraph)

# Vo Huong

edges <- c(1,2, 3,2, 2,4)

g<-graph(edges, n=max(edges), directed=FALSE)

plot(g)

# 13. Cài đặt đồ thị có hướng:

## a. Python:

import pandas as pd

import networkx as nx

import matplotlib.pyplot as plt

df = pd.DataFrame({'from': ['A', 'B', 'C', 'D'], 'to': ['B', 'D', 'A', 'A']})

G = nx.from\_pandas\_edgelist(df, 'from', 'to', create\_using=nx.Graph())

nx.draw(G, with\_labels=True, node\_size=1500, alpha=0.3, arrows=True)

plt.title("UN-Directed")

plt.show()

## b. R:

library(igraph)

a = make\_graph(c(1, 2, 2, 3, 3, 4, 5, 6, 4,1, 2,4, 4,5, 6,1), directed = TRUE)

plot(a)

# 14. Cài đặt thuật toán sắp xếp chọn:

## a. Python:

def selection\_sort(input\_list):

for idx in range(len(input\_list)):

min\_idx = idx

for j in range( idx +1, len(input\_list)):

if input\_list[min\_idx] > input\_list[j]:

min\_idx = j

input\_list[idx], input\_list[min\_idx] = input\_list[min\_idx], input\_list[idx]

l = [19,2,31,45,30,11,121,27]

selection\_sort(l)

print(l)

## b. R:

vec <- sample(1:100)

sort <- function(x) {

n<- length(x)

for (i in 1:(n-1)) {

for (j in (i+1):n) {

if(x[j] < x[i]) {

temp <-x[i]

x[i] <- x[j]

x[j] <- temp

}

}

}

return(x)

}

sort(vec)

selfsort <- function(x) {

if (length(x)>1) {

min <- which.min(x)

c(x[min], selfsort(x[-min]))

} else x

}

selfsort(vec)

# 15. Cài đặt thuật toán sắp xếp chèn:

## a. Python:

def insertion\_sort(InputList):

for i in range(1, len(InputList)):

j = i - 1

nxt\_element = InputList[i]

while (InputList[j] > nxt\_element) and (j >= 0):

InputList[j + 1] = InputList[j]

j = j - 1

InputList[j + 1] = nxt\_element

list = [19, 2, 31, 45, 30, 11, 121, 27]

insertion\_sort(list)

print(list)

## b. R:

insertionsort\_function <- function(A){

for (j in 2:length(A)) {

key = A[j]

# insert A[j] into sorted sequence A[1,...,j-1]

i = j - 1

while (i > 0 && A[i] > key) {

A[(i + 1)] = A[i]

i = i - 1

}

A[(i + 1)] = key

}

A

}

insertionsort\_function(c(5, 2, 4, 6, 1, 3))

# 16. Cài đặt thuật toán sắp xếp nổi bọt:

## a. Python:

def bubblesort(list):

for iter\_num in range(len(list)-1,0,-1):

for i in range(iter\_num):

if list[i]>list[i+1]:

temp = list[i]

list[i] = list[i+1]

list[i+1] = temp

list = [19,2,31,45,6,11,121,27]

bubblesort(list)

print(list)

## b. R:

vec = c(1,-1,3,2,10,9)

bubble <- function(x){

n<-length(x)

for(j in 1:(n-1)){

for(i in 1:(n-j)){

if(x[i]>x[i+1]){

temp<-x[i]

x[i]<-x[i+1]

x[i+1]<-temp

}

}

}

return(x)

}

bubble(vec)

# 17. Cài đặt thuật toán sắp xếp nhanh - quick sort:

## a. Python:

def partition(arr, low, high):

i = (low - 1)

pivot = arr[high]

for j in range(low, high):

if arr[j] <= pivot:

i = i + 1

arr[i], arr[j] = arr[j], arr[i]

arr[i + 1], arr[high] = arr[high], arr[i + 1]

return (i + 1)

def quickSort(arr, low, high):

if len(arr) == 1:

return arr

if low < high:

pi = partition(arr, low, high)

quickSort(arr, low, pi - 1)

quickSort(arr, pi + 1, high)

arr = [10, 7, 8, 9, 1, 5]

n = len(arr)

quickSort(arr, 0, n - 1)

print("Sorted array is:")

for i in range(n):

print("%d" % arr[i])

## b. R:

vec = c(1,-2,3,6,2,1,9)

quickSort <- function(arr) {

mid <- sample(arr, 1)

left <- c()

right <- c()

lapply(arr[arr != mid], function(d) {

if (d < mid) {

left <<- c(left, d)

}

else {

right <<- c(right, d)

}

})

if (length(left) > 1) {

left <- quickSort(left)

}

if (length(right) > 1) {

right <- quickSort(right)

}

c(left, mid, right)

}

quickSort(vec)

# 18. Cài đặt thuật toán heap sort:

## a. Python:

def heapify(arr, n, i):

largest = i

l = 2 \* i + 1

r = 2 \* i + 2

if l < n and arr[i] < arr[l]:

largest = l

if r < n and arr[largest] < arr[r]:

largest = r

if largest != i:

arr[i], arr[largest] = arr[largest], arr[i]

heapify(arr, n, largest)

def heapSort(arr):

n = len(arr)

for i in range(n // 2 - 1, -1, -1):

heapify(arr, n, i)

for i in range(n - 1, 0, -1):

arr[i], arr[0] = arr[0], arr[i]

heapify(arr, i, 0)

arr = [12, 11, 13, 5, 6, 7]

heapSort(arr)

n = len(arr)

print("Sorted array is")

for i in range(n):

print("%d" % arr[i])

## b. R:

heap.building<-function(vec)

{

len=length(vec)

heap=vec

for (j in len:1)

{

heap=modify.heap(heap,j)

}

return(heap)

}

is.heap<-function(heap,root\_i)

{

i=root\_i

res=T

while(2\*i<=length(heap)&res)

{

son=c(heap[2\*i],heap[2\*i+1])

son=son[!is.na(son)]

res=all(heap[i]<=son)

i=i+1

}

return(res)

}

modify.heap<-function(heap,root\_i)

{

len=length(heap)

flag=1

while (root\_i\*2<=len&&flag==1)

{

left\_i=root\_i\*2

right\_i=root\_i\*2+1

flag=0

son=c(heap[left\_i],heap[right\_i])

son=son[!is.na(son)]

min\_ind=which.min(son)

if (heap[root\_i]>son[min\_ind])

{

flag=1

heap\_ind=c(left\_i,right\_i)[min\_ind]

tmp=heap[heap\_ind]

heap[heap\_ind]=heap[root\_i]

heap[root\_i]=tmp

root\_i=heap\_ind

}

}

return(heap)

}

heap.sort<-function(heap)

{

sorted=NULL

len=length(heap)

while(len>0)

{

sorted=c(sorted,heap[1])

len=length(heap)

heap[1]=heap[len]

heap=heap[1:(len-1)]

heap=modify.heap(heap,root\_i=1)

len=len-1

}

return(sorted)

}

vec = c(-1,2,1,10,3,9)

heap=heap.building(vec)

heap\_sort=heap.sort(heap)

heap\_sort

# 19. Cài đặt thuật toán sắp xếp trộn – merge sort:

## a. Python:

def merge\_sort(unsorted\_list):

if len(unsorted\_list) <= 1:

return unsorted\_list

middle = len(unsorted\_list) // 2

left\_list = unsorted\_list[:middle]

right\_list = unsorted\_list[middle:]

left\_list = merge\_sort(left\_list)

right\_list = merge\_sort(right\_list)

return list(merge(left\_list, right\_list))

def merge(left\_half,right\_half):

res = []

while len(left\_half) != 0 and len(right\_half) != 0:

if left\_half[0] < right\_half[0]:

res.append(left\_half[0])

left\_half.remove(left\_half[0])

else:

res.append(right\_half[0])

right\_half.remove(right\_half[0])

if len(left\_half) == 0:

res = res + right\_half

else:

res = res + left\_half

return res

unsorted\_list = [64, 34, 25, 12, 22, 11, 90]

print(merge\_sort(unsorted\_list))

## b. R:

merge<-function(a,b) {

r<-numeric(length(a)+length(b))

ai<-1; bi<-1; j<-1;

for(j in 1:length(r)) {

if((ai<=length(a) && a[ai]<b[bi]) || bi>length(b)) {

r[j] <- a[ai]

ai <- ai+1

} else {

r[j] <- b[bi]

bi <- bi+1

}

}

r

}

mergesort<-function(A) {

if(length(A)>1) {

q <- ceiling(length(A)/2)

a <- mergesort(A[1:q])

b <- mergesort(A[(q+1):length(A)])

merge(a,b)

} else {

A

}

}

x<- c(-1,2,1,10,3,9,234,123,543,324)

mergesort(x)