逢甲大學

資訊工程學系

計算機演算法

作業一

分析Sorting 時間

授課老師：陳清文

系 級：自控四甲

學 號：D0282187

姓 名：楊竑昕

座 號：64

# ㄈ做法

建立一個新物件RandomInt 內有：

* 建構子RandomInt.new(number:亂數個數,range:亂數範圍)，產生整數亂數陣列。
* methods :
  + bubble\_sort
  + selection\_sort
  + insertion\_sort
  + quick\_sort
  + heap\_sort.

建立一個新物件MinHeap內有：

* 建構子MinHeap.new(array)，產生min\_heap。
* methods :
  + size
  + add(number)
  + pop
  + empty?

主程式，詢問6次IO輸入建立對應的6個RandomInt物件，其亂數個數為方才輸入的個數。

6個RandomInt物件的每個排序方式各呼叫25次，並將時間記錄下來。

# 程式碼

排版好的程式碼在Github上：<https://github.com/jk195417/sorting_algorithm_compare>

require\_relative './min\_heap'

class RandomInt

attr\_accessor :array

attr\_reader :number

def initialize(number: 10, range: 100)

prng = Random.new

@array = 1.upto(number).map { prng.rand(range) }

@number = number

end

def bubble\_sort

result = Array.new(@array)

loop do

swapped = false

(result.length - 1).times do |i|

next unless result[i] > result[i + 1]

result[i], result[i + 1] = result[i + 1], result[i] # Swap

swapped = true

end

break unless swapped

end

result

end

def selection\_sort

result = Array.new(@array)

0.upto(result.length - 1) do |i|

min = i

(i + 1).upto(result.length - 1) do |j|

min = j if result[j] < result[min]

end

result[i], result[min] = result[min], result[i] # Swap

end

result

end

def insertion\_sort

result = Array.new(@array)

return result if result.length < 2

(1..result.length - 1).each do |i|

value = result[i]

j = i - 1

while (j >= 0) && result[j] > value

result[j + 1] = result[j]

j -= 1

end

result[j + 1] = value

end

result

end

def quick\_sort

result = Array.new(@array)

bottom = []

top = []

top[0] = 0

bottom[0] = result.length

i = 0

while i >= 0

l = top[i]

r = bottom[i] - 1

if l < r

pivot = result[l]

while l < r

r -= 1 while result[r] >= pivot && l < r

if l < r

result[l] = result[r]

l += 1

end

l += 1 while result[l] <= pivot && l < r

if l < r

result[r] = result[l]

r -= 1

end

end

result[l] = pivot

top[i + 1] = l + 1

bottom[i + 1] = bottom[i]

bottom[i] = l

i += 1

else

i -= 1

end

end

result

end

def heap\_sort

result = []

heap = MinHeap.new(Array.new(@array))

result << heap.pop until heap.empty?

result

end

end

class MinHeap

attr\_accessor :array

def initialize(ary = [])

# heap array index start at 1

@array = [0]

ary.each { |a| add a }

end

def size

@array.length - 1

end

def add(n)

@array << n

node = size

parent = (node.odd? ? (node - 1) / 2 : node / 2)

while @array[node] < @array[parent]

@array[node], @array[parent] = @array[parent], @array[node]

node = parent

parent = (parent.odd? ? (parent - 1) / 2 : parent / 2)

break if parent == 0

end

end

def pop

if empty?

result = nil

else

@array[1], @array[size] = @array[size], @array[1]

result = @array.pop

parent = 1

left = parent \* 2

right = left + 1

while @array[left]

node = if @array[right] && @array[right] < @array[left]

right

else

left

end

if @array[parent] > @array[node]

@array[parent], @array[node] = @array[node], @array[parent]

end

parent = node

left = parent \* 2

right = left + 1

end

end

result

end

def empty?

size <= 0 ? true : false

end

end

require 'benchmark'

require\_relative './random\_int'

threads = []

cases = []

6.times do

print 'please enter a integer : '

n = gets.chomp!

n = Integer(n)

cases << RandomInt.new(number: n)

end

puts "\nstart sorting..."

cases.each do |c|

puts "number = #{c.number}"

Benchmark.bm(20) do |x|

x.report('bubble\_sort') { 25.times { c.bubble\_sort } }

x.report('selection\_sort') { 25.times { c.selection\_sort } }

x.report('insertion\_sort') { 25.times { c.insertion\_sort } }

x.report('quick\_sort') { 25.times { c.quick\_sort } }

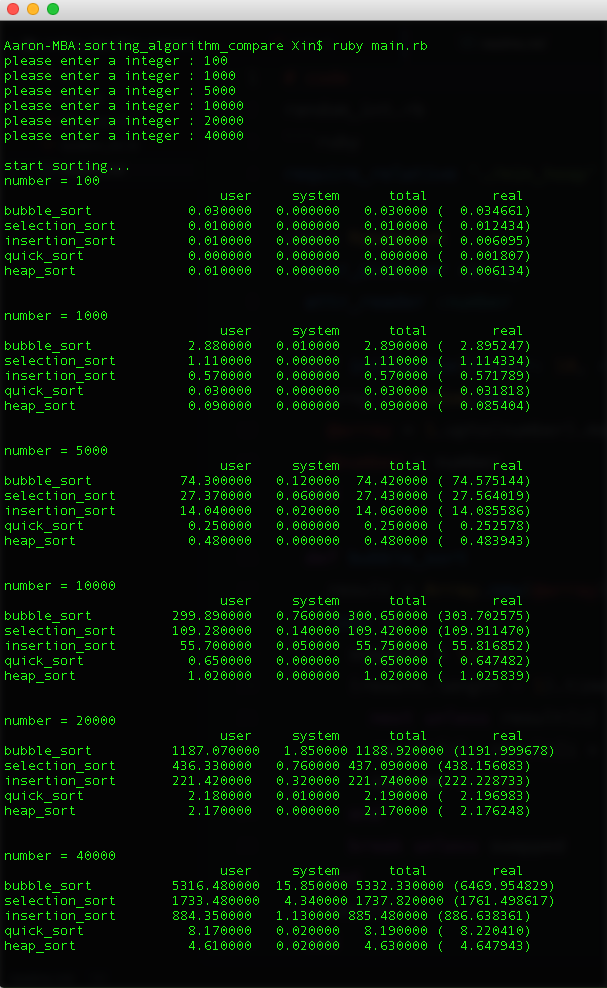
x.report('heap\_sort') { 25.times { c.heap\_sort } }

end

puts "\n" \* 2

end

# 執行結果



# 討論與心得

實作這些排序方法印象很深刻，好的方法處理大一點的數據，可以省下非常多的時間，以後寫程式會特別注意自己寫出來的時間複雜度，精進自己的產品效能。

這次等好久，而且我才輸入到40000個，這次的作業後，讓我萌生想換電腦的念頭。