Алгоритми та структури даних

Лабораторна робота 2: Алгоритми сортування

Варіант 6

Протокол

Михайло Корешков
ФІ-81

Фізико-Технічний Інститут
НТУУ ”КПІ ім. Ігоря Сікорського”
2020

Зміст

1 Структура

```
/sorting.jl                # Основний файл із кодом
- function bubblesort!(v::Vector{T})
    Сортування вектору v алгоритмом бульбашки.
    Приймає вектори будь-якого типу, який підтримує порівняння
    Повертає кортеж (cmps, swaps)

- function quicksort!(v::Vector{T}, pretty=false, pivoter::Function=((v, lo,
↪  hi)->fld(lo+hi, 2)))
    Сортування вектору v алгоритмом швидкого сортування
    Приймає вектор (аналогічно bubblesort!), флаг для увімкнення покрокового
     ↪  виводу стану алгоритма,
      та функцію, що обирає pivot-елемент підмасиву. За замовчуванням,
       ↪  обирається серединний
      елемент підмасиву
    Повертає кортеж (cmps, swaps)

  - function divide!(v::Vector{T}, lo, hi, pivoter::Function)
      ”Worker”-функція, що власне займається вибором pivot-елементу та обміном
       ↪  інверсій.
      Приймає весь(!) сортований вектор, індекси початку та кінця підмасиву,
       ↪  який необхідно
        відсортувати, та функцію, що обирає pivot-елемент
      Повертає індекс точки нового розбиття
  - function divide_pretty!
      Аналогічно divide!, але кожну ітерацію виводить в stdout інформацію про
       ↪  стан алгоритма

- function generate_report()
    Функція для тестування швидкодії алгоритмів. Запускає обидва алгоритма
     ↪  Samples разів для
      кожної довжини випадкового вхідного вектору з списку evenNs, а потім
       ↪  oddNs
    Результат записується в файл mk-sorting-report.csv

    - function report_algo(label, Ns, Samples, algo::Function)
        Функція, що виконує всю роботу. Приймає назву тесту, список розмірів
         ↪  вхідного вектору,
          кількість повторів кожного підтесту та власне саму функцію, що
           ↪  реалізує алгоритм.
        Повертає DataFrame із даними про кількість тестів та кількостями
         ↪  обмінів і порівнянь
          для кожного тесту
```

2 Проектні рішення

2.1 Алгоритм швидкого сортування

Я вирішив обрати не-рекурсивний підход із використанням додаткового стеку ”задач” - вектор кортежів типу (Int, Int), які містять індекси початку та кінця підмасиву, який потрібно відсортувати наступним. Перше, що відбувається у основному циклі - зі стеку виймається задача. Цикл продовжується доки стек задач не пустий.

3 Аналіз швидкодії

Графіки залежності кількості операцій від розміру масива. Масштаб log2-log2. Окремо для парних та непарних N.

4 Висновки

Основний висновок - Quicksort значно швидший за Bubblesort, але тільки починаючи з деякого значення N. Для N=1000 кількість обмінів Quicksort у більш ніж 1000 разів менша за таку в Bubblesort.

4.1 Вихідні дані порівняння

```
Label,N,Cmps,Swaps
bubble-odd,13,122,39
bubble-odd,26,523,163
bubble-odd,52,2272,670
bubble-odd,104,9530,2699
bubble-odd,208,39704,10736
bubble-odd,416,162738,43113
bubble-odd,832,662847,173263
bubble-odd,1664,2684015,693900
bubble-odd,3328,10833643,2768631
bubble-odd,6656,43557041,11062716
bubble-odd,13312,175069999,44309403
bubble-even,10,67,23
bubble-even,20,308,95
bubble-even,40,1315,396
bubble-even,80,5514,1582
bubble-even,160,23008,6362
bubble-even,320,95144,25725
bubble-even,640,388716,102487
bubble-even,1280,1587788,409949
bubble-even,2560,6389771,1633407
bubble-even,5120,25730602,6557728
bubble-even,10240,103562979,26210419
quick-odd,13,144,11
quick-odd,26,341,28
quick-odd,52,788,70
quick-odd,104,1807,170
quick-odd,208,4125,403
quick-odd,416,9068,919
```

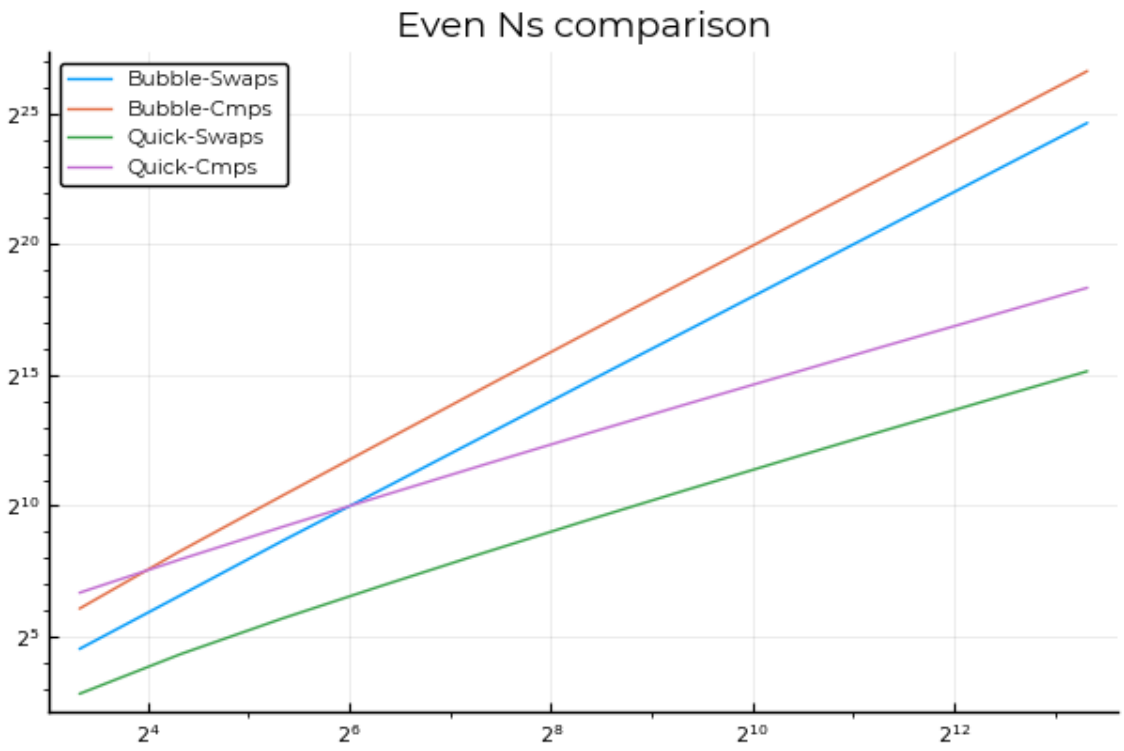

Figure 1: Парні N

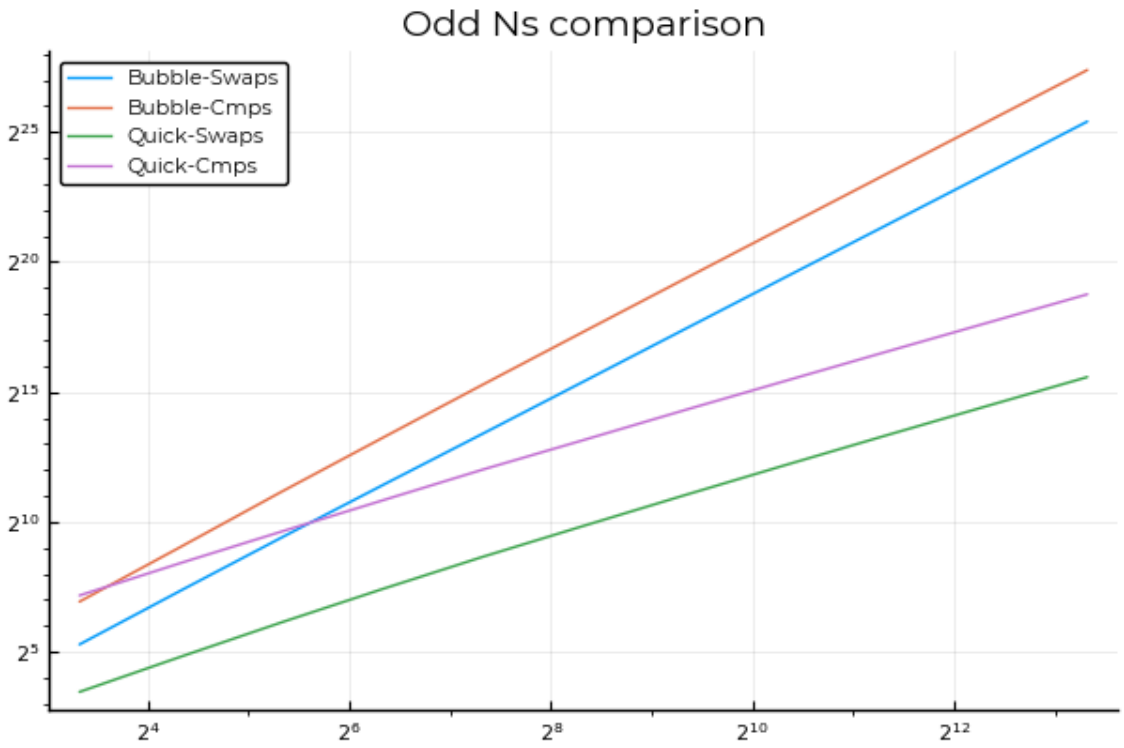

Figure 2: Непарні N

```
quick-odd,832,20088,2079
quick-odd,1664,43805,4641
quick-odd,3328,95179,10243
quick-odd,6656,205589,22408
quick-odd,13312,442029,48669
quick-even,10,102,7
quick-even,20,247,20
quick-even,40,579,51
quick-even,80,1328,123
quick-even,160,3009,291
quick-even,320,6733,675
quick-even,640,14888,1531
quick-even,1280,32491,3434
quick-even,2560,71160,7602
quick-even,5120,153246,16658
quick-even,10240,331443,36258
```

5 Код

```
#=
# Варіант 6
# 1. Bubble sort
# 2. Quicksort
=#
using Random
using DataFrames
using CSV

function bubblesort!(v::Vector{T}) where T
    len = length(v)
#     println("init: ", v)

    cmps = 0
    swaps = 0

    for i in 1:len
        sorted::Bool = true
        for j in 1:len-1
            if v[j] > v[j + 1]
                v[j],v[j + 1] = v[j + 1],v[j] # Swap
                swaps+=1
                sorted = false
            end
            cmps+=1 # At least 1 comparison at the if
        end
        sorted && break
    end

#     println("sorted: ", v)

    return (cmps, swaps)
```

```
end

function quicksort!(v::Vector{T}, pretty=false, pivoter::Function=((v, lo,
↪  hi)->fld(lo+hi, 2))) where T
    N = 0
    function print_v_frame(v, p, lo, hi)
        len = length(v)
        println(N)
        N+=1
        println(v)
        print(" ")
        for i=1:len
            if     i==lo print("[  ")
            elseif i==p  print("^  ")
            elseif i==hi print("]  ")
            else print("   ") end
        end
        println()
    end

    cmps = 0
    swaps = 0

    # Pretty version
    # Divide and calculate new pivot point
    function divide_pretty!(v::Vector{T}, lo, hi, pivoter::Function)
        p = pivoter(v, lo, hi)
        pivot = v[p] # Select new pivot - val of middle element
        print_v_frame(v, p, lo, hi)
        while true
            # Find first inversion
            while v[lo] < pivot lo+=1; cmps+=1; end; cmps+=1;
            print_v_frame(v, p, lo, hi)
            while v[hi] > pivot hi-=1; cmps+=1; end; cmps+=1;
            print_v_frame(v, p, lo, hi)
            # If sorted: no inversions found and frame shrinked to 0 width,
            ↪  return pivot point
            cmps+=1
            lo ≥ hi && (println("lo>=hi"); return hi)
            # otherwise, swap inversion and continue
            v[lo], v[hi] = v[hi], v[lo]
            swaps+=1

            print_v_frame(v, p, lo, hi)
        end
        print_v_frame(v, p, lo, hi)
    end
    # Normal version
    function divide!(v::Vector{T}, lo, hi, pivoter::Function)
        p = pivoter(v, lo, hi)
        pivot = v[p] # Select new pivot - val of middle element
        while true
```

```
            # Find first inversion
            while v[lo] < pivot lo+=1; cmps+=1; end; cmps+=1;
            while v[hi] > pivot hi-=1; cmps+=1; end; cmps+=1;
            # If sorted: no inversions found and frame shrinked to 0 width,
             ↪  return pivot point
            cmps+=1
            lo ≥ hi && return hi
            # otherwise, swap inversion and continue
            v[lo], v[hi] = v[hi], v[lo]
            swaps+=1
        end
    end

    Task = Tuple{Int, Int}
    tasks::Vector{Task} = []

    # Initial task
    push!(tasks, (1, length(v)))
    divider = pretty ? divide_pretty! : divide!;

    while !isempty(tasks)
        lo, hi = pop!(tasks) # Get a task
        cmps+=1
        lo < hi || continue
        p = divider(v, lo, hi, pivoter) # Calculate pivot point index
        push!(tasks, (lo,p))    # Add new task for left
        push!(tasks, (p+1,hi))  #   and right partition
    end

    return (cmps, swaps)
end

function benchmark(algo::Function, vlen::Int, samples::Int, args...)
    # cmps, swaps
    stats = [0, 0]
    @time begin
        v = [1:vlen;]
        for i in 1:samples
            shuffle!(v)
            stats .+= algo(v, args...)
        end
    end
    stats = stats ./ samples
    println("avg cmps: ", stats[1], "     avg swaps: ", stats[2])
end

function generate_report()
    Samples = 100

    evenNs = ((n)->10*(2^n)).([0:10;])
    oddNs = ((n)->13*(2^n)).([0:10;])

```

```
    function report_algo(label, Ns, Samples, algo::Function)
        local df = DataFrame(Label = String[], N = Int[], Cmps = Int[], Swaps =
        ↪ Int[])
        for N in Ns
            v = [1:N;]
            stats = [0, 0]
            println(label, " ", N, "...")
            for i in 1:Samples
                shuffle!(v)
                stats .+= algo(v)
            end
            stats = fld.(stats,Samples)
            push!(df, ( label, N, stats[1], stats[2] ))
        end
        return df
    end

    df = report_algo("bubble-odd", oddNs, Samples, bubblesort!)
    append!(df, report_algo("bubble-even", evenNs, Samples, bubblesort!))
    append!(df, report_algo("quick-odd", oddNs, Samples, quicksort!))
    append!(df, report_algo("quick-even", evenNs, Samples, quicksort!))

    CSV.write("mk-sorting-report.csv", df)
end
```