Самостійна робота

з предмету Проектування та аналіз обчислювальних алгоритмів

студента групи ІСзп 71
Бутузова О. В.

Варіант №4

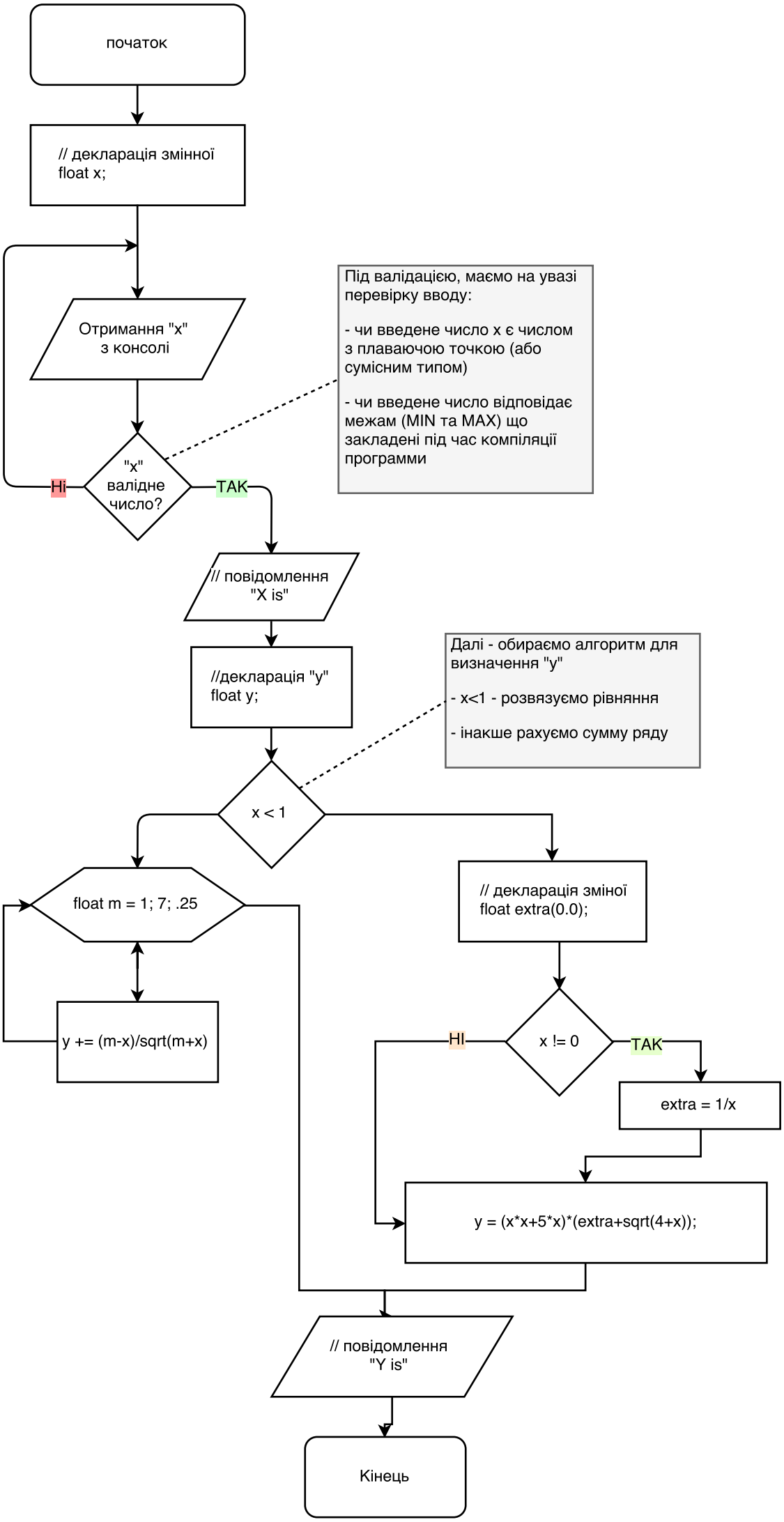
Завдання №1

Спроектувати алгоритм розв'язку задачі використовуючи базові алгоритмічні конструкції, реалізувати мовою програмування C/C++. У звіті вказати, яким вимогам повинен задовольняти розв'язок задачі, вхідні дані та результати, представити блок-схему алгоритму (або псевдокод, або діаграму дії, тощо), текст програмного коду.

$$y = \begin{cases} (x^2 + 5x)(\dfrac{1}{x} + \sqrt{x+4}), & \text{якщо} \quad x < 1 \\ \sum_{m=1}^{7} \dfrac{m - x}{\sqrt{m + x}} & \text{якщо} \quad x \geq 1 \end{cases} \quad \begin{array}{c} x \in [-1,5; 3,5] \\ \text{крок } h = 0.25 \end{array}$$

За умовами задачі створено алгоритм (дивіться представлення алгоритму на насутпній сторінці) та реалізацію программи мовою C++. Данна программа компілюється і запускається з вихідного файлу *problem_1.cpp* наступним чином:

```
c++ problem_1.cpp -o algorithm -std=c++11 && ./algorithm
```


початок
// декларація змінної
float x;
Отримання "x"
з консолі
"x"
валідне
число?
Ні
ТАК
Під валідацією, маємо на увазі перевірку вводу:
- чи введене число x є числом з плаваючою точкою (або сумісним типом)
- чи введене число відповідає межам (MIN та MAX) що закладені під час компіляції программи
// повідомлення
"X is"
//декларація "y"
float y;
Далі - обираємо алгоритм для визначення "y"
- x<1 - розвязуємо рівняння
- інакше рахуємо сумму ряду
x < 1
float m = 1; 7; .25
y += (m-x)/sqrt(m+x)
// декларація зміної
float extra(0.0);
x != 0
Ні
ТАК
extra = 1/x
y = (x*x+5*x)*(extra+sqrt(4+x));
// повідомлення
"Y is"
Кінець

Завдання №2

Завдання: Проілюструйте процес упорядкування послідовності чисел ([20, 4, 45, 1, 2, 6, 88]) алгоритмом сортування вставкою:

1. Перевірка довжини масиву
2. Якщо довжина менше або дорівнює 1 - преривання функції сортування твердженням "return"
3. Початок ітерації елемента масиву за індексом “1” (array[1]) що має значення “4”
4. Значення “20” елемента масиву за індексом 0 (array[0]) скопійовано в елемент масиву за індексом 1 (array[1])
5. Елемента масиву за індексом 0 (array[0]) набув нового значення “4” (найменше число данної ітерації)
6. Початок ітерації елемента масиву за індексом “2” (array[2]) що має значення “45”
7. Елемента масиву за індексом 2 (array[2]) не потребує нового значення оскільки є найбішим значенням сортування цієї ітерації.
8. Початок ітерації елемента масиву за індексом “3” (array[3]) що має значення “1”
9. Значення “45” елемента масиву за індексом 2 (array[2]) скопійовано в елемент масиву за індексом 3 (array[3])
10. Значення “20” елемента масиву за індексом 1 (array[1]) скопійовано в елемент масиву за індексом 2 (array[2])
11. Значення “4” елемента масиву за індексом 0 (array[0]) скопійовано в елемент масиву за індексом 1 (array[1])
12. Елемента масиву за індексом 0 (array[0]) набув нового значення “1” (найменше число данної ітерації)
13. Початок ітерації елемента масиву за індексом “4” (array[4]) що має значення “2”
14. Значення “45” елемента масиву за індексом 3 (array[3]) скопійовано в елемент масиву за індексом 4 (array[4])

15. Значення “20” елемента масиву за індексом 2 (array[2]) скопійовано в елемент масиву за індексом 3 (array[3])
16. Значення “4” елемента масиву за індексом 1 (array[1]) скопійовано в елемент масиву за індексом 2 (array[2])
17. Елемента масиву за індексом 1 (array[1]) набув нового значення “2” (найменше число данної ітерації)
18. Початок ітерації елемента масиву за індексом “5” (array[5]) що має значення “6”
19. Значення “45” елемента масиву за індексом 4 (array[4]) скопійовано в елемент масиву за індексом 5 (array[5])
20. Значення “20” елемента масиву за індексом 3 (array[3]) скопійовано в елемент масиву за індексом 4 (array[4])
21. Елемента масиву за індексом 3 (array[3]) набув нового значення “6” (найменше число данної ітерації)
22. Початок ітерації елемента масиву за індексом “6” (array[6]) що має значення “88”
23. Елемента масиву за індексом 6 (array[6]) не потребує нового значення оскільки є найбішим значенням сортування цієї ітерації.
24. Кінець процедури сортування.

Додатково

- В файлі code.cpp подано дві імплементацію алгоритму сортування вставкою (array_insertion_sort та array_insertion_sort_inplace_swap).
- Приведено приклад реалізації сортування (який можна скомпілювати та запустити насутпним чином):
  c++ problem_2_sort.cpp code.cpp -o sort -std=c++11 && ./sort

Завдання №3

Завдання Для послідовності цілих чисел $x[1], \ldots, x[n]$. Знайти максимальну довжину її зростаючої підпослідовності (число дій порядку $n \log n$).

Цю задачу можна імплементувати n ітерацій, і відслідковуючи довжину найдовшої послідовності. Приклад імплементації подано у файлі code.cpp, приклад використання в problem_3_lciss.cpp, данний приклад можна скомпілювати за запустити за допомогою команд оболонки bash

```
c++ problem_3_lciss.cpp code.cpp -o lciss -std=c++11 && ./lciss
```

Завдання №4

Завдання: У впорядкованому масиві цілих чисел ai , i=1...n знайти номер елемента с використовуючи метод двійкового пошуку. Передбачається, що цей елемент знаходиться в масиві.

Алгоритм бінарного пошуку, являє собою швидкий ($O(\log n)$) алгоритм пошуку, мінусом данного алгоритму є те що працює він лише на попередньо відсортованих масивах. Алгоритм роботи полягає в наступному:

1. Визначаються початковий і останній елемент масива для пошука.
2. Починаємо цикл з умовою що допоки початок не дорівнюватиме кінцю ми будемо ітерувати тіло циклу.
3. Визначимо центр пошукового діапазону, додавши що "початку середнє різниці між кінцем та початком.
4. Якщо Елемент масиву дорівнює тому значенню що ми шукаємо, робота алгоритму повертається і ми повертає індекс знайденого елементу, або логічну змінну що елемент знайдено.
5. Якщо Елемент масиву менше того значення що ми шукаємо, логічно уявити що шукане значення знаходить в правій стороні, в такому разі “початком” діапазону пошуку стає наступний за порівнювальним елемент масиву (middle+1).
6. Якщо Елемент масиву більше того значення що ми шукаємо, логічно уявити що шукане значення знаходить в лівій стороні, в такому разі “кінцем” діапазону пошуку стає попередній порівнювальним елемент масиву (middle).
7. Якщо цикл закінчився без результатно то повертається значення -1, що означає щ індекс не знайдено.

Імплементація розташована в code.cpp, реалізацію і приклад використання в problem_4_binary_search.cpp, данний приклад можна скомпілювати за запустит за допомогою команд оболонки bash

```
c++ problem_4_binary_search.cpp code.cpp -o bs -std=c++11 && ./bs
```

Завдання №5

Завдання з загального файлу завдань було замінене на наступне за згодою викладача

Завдання: Заповнити довільно великий масив випадковими числовими данними і відсортувати його кількома (2 два ніби замало, а три щоб різна complexity не набереться) різними алгоритмами сортуваня з різною складністю виконання ($O(n \log n)$, $O(n^2)$)

Протестувати усі імплементовані алгоритми і написати висновки щодо часу і простороу складності алгоритмів.

Імплементовані алгоритми сортування:

1. Сортування мотодом Бульбашок
2. Сортування Вибіркою
3. Сортування Вставкою
4. Сортування Вставкою (з заміною елементів)[1]
5. Сортування Злиттям
6. Швидке сортування

Тестувались алгоритми сортування наступним чином:

1. Генерувалось 100 масивів випадкових чисел різною довжиною n (2^8, 2^{10}, 2^{16} і 2^{21}).
2. Программа сортувальник (problem_5_sort.cpp) отримувала на вході алгоритм що підлягав тестуванню і сам масив.
3. Час що витрачався на сортування замірявся (за допомогою timer.c) на вході і виході, різниця записувалась в журнал.
4. По-кожном з алгоритмів сортування вираховувалось середнє значення від якого віднімалось середнє значення ініціалізації программи та отримання данних.

[1] демонстраційний зразок змін які відбуваються з швидкодією коли замість замість однієї заміни ввести зміну обох зміниих

Інструменти:

1. Сортувальник (problem_5_sort.cpp) программа що отримувала на вході один з попередньо імплементованих алгоритмів сортування та масц з цілих числе як аргумент. Программа не виводила абсолютно нічого і займалась лише там що запускала механізм сортування для отриманого масиву.

   Компіляція:

   c++ problem_5_sort.cpp code.cpp -o sorter -std=c++11

2. Таймер (timer.c) - повертав системний час у форматі unixtimestamp і мікросекунд.

   Компіляція:

   c++ timer.cpp -o timer -std=c++11

3. Генератор довільного масиву problem_5_generate.cpp, программа, що просто генерує массив довільних значень.

   Компіляція:

   c++ problem_5_generate.cpp code.cpp -o generate -std=c++11

4. Генерація тестових данних - скрипт (tests.sh) написаний для Bourne again shell, що генерує тестові масиви, запускає тестування імплементованих алгоритмів сортування та протоколює метрики.

5. Аналізатор метрик (report.py) - скрипт написаний на Python3 що читає наші рапорти з метриками і видає на їх основі LaTeXсумісну відповідь.

Таблиця результатів:

Алгоритми	Розмір масиву цілих чисел			
	2^8	2^{16}	2^{16}	2^{21}
QuickSort	0.0003	0.0001	0.0109	0.3572
MergeSort	0.0004	0.0001	0.0325	0.9553
Insertion	0.0004	0.0009	0.3594	11.6863
Selection	0.0006	0.0017	0.5029	15.9639
Insertion*	0.0004	0.0030	1.0228	33.7722
BubbleSort	0.0005	0.0053	1.6633	53.1013

Таблиця показу середню швидкість в секундах на сортування (за мінусом середнього часу на ініціалізацію).

Висновки:

- На невеликих масивих інколи краще використовувати алгоритми з складністю $O(n^2)$ - оскільки в данному випадку приріст швидкодії відбувається саме через простоту алгоритму сортуваннь наприклад на масивах розміром 2^4 InsertionSort побив QuickSort (результат проміжних тестів).
- Лінійні (відсутні у данному наборі), лінійно логарифмічні та квадратичні алгоритми седуть себе відповідно до своїх класифікацій у відношенні до збільшення/або зменшення обєму вхідних данних - але це ріст/спа д неє чітко квадратичним (окрім як у випадку бульбашкового сортування) чи лінійно-логарифмічним і загалом залежить від того як саме розташовані алементи в масиві. Іншими словами - "найгірший" випадок зустрічається вкрай рідко.
- Алгоритми одного класу складності різняттся між собою доволі значно, тому слід уважно обирати алгоритм сортування зважаючи на обєм вхідних данних.
- Має місце імплементації складніших алгоритмів сортування що зважать на розмір, тип вхідних данних, наприклад у відповідності від розміру данних використовувати різні алгоритми щ гарантуватимуть більш швидкий результ.
- Існують різні імплементації спец алгоритмів що працюють на спеціальних структурах данних, що гарантують не тільки швидке сортування але і швидкий пошук і видалення з структури (напраклад двійкові дерева тощо).