Звіт з лабораторної роботи
за дисципліною "алгоритми і структури данних"
студента групи ПА-17-1
Панасенка Егора Сергійовича
Кафедра комп’ютерних технологій, фпм, дну
2017/2018 навч.р.

1. Постановка задачі:

   Розробити дві програми на мові C/C++, які реалізують два алгоритми сортування масивів:

   1. за методом «поганого» сортування; (Сортування бульбашкою)
   2. за методом «ефективного» сортування. (Сортування злиттям)

   Вихідний масив – масив цілих чисел. Вводити із файлу.

   Результатний відсортований масив записати в інший файл.

   Розглянути декілька тестових прикладів з різними розмірами масивів. Програмно визначити час роботи кожного з алгоритмів на кожному з тестових прикладів з різними розмірами масивів. Результати навести в табличному вигляді.

   Побудувати графіки залежності часу роботи алгоритмів від розміру масивів.

   Зробити порівняльний аналіз, навести висновки.

2. Опис алгоритмів:

   1. Сортування бульбашкою

Клас	Алгоритм сортування
Структура даних	Масив
Найгірша швидкодія	$O(n^2)$
Найкраща швидкодія	$O(n)$
Середня швидкодія	$O(n^2)$
Просторова складність у найгіршому випадку	$O(n)$ загальний, $O(1)$ допоміжний
Оптимальний	Ні

      1. Сортування обміном або сортування бульбашкою є простим алгоритмом сортування. Алгоритм працює таким чином – у поданому наборі даних (списку чи масиві) порівнюються два сусідні елементи. Якщо один з елементів, не відповідає

критерію сортування (є більшим, або ж, навпаки, меншим за свого сусіда), то ці два елементи міняються місцями. Прохід по списку продовжується до тих пір, доки дані не будуть відсортованими. Алгоритм отримав свою назву від того, що процес сортування за ним нагадує поведінку бульбашок повітря у резервуарі з водою. Оскільки для роботи з елементами масиву він використовує лише порівняння, це сортування на основі порівнянь.

2. Продуктивність: Складність алгоритму у найгіршому у середньостатистичному випадку рівна $O(n^2)$, де n — кількість елементів для сортування. Існує чимало значно ефективніших алгоритмів, наприклад, з найгіршою ефективністю рівною O(n log n). Тому даний алгоритм має низьку ефективність у випадках, коли N є досить великим, за винятком рідкісних конкретних випадків, коли заздалегідь відомо, що масив з самого початку буде добре відсортований.

2. Сортування злиттям

   Сортування злиттям (англ. merge sort) — алгоритм сортування, в основі якого лежить принцип «Розділяй та володарюй».

   В основі цього способу сортування лежить злиття двох упорядкованих ділянок масиву в одну впорядковану ділянку іншого масиву. Злиття двох упорядкованих послідовностей можна порівняти з перебудовою двох колон солдатів, вишикуваних за зростом, в одну, де вони також розташовуються за зростом. Якщо цим процесом керує офіцер, то він порівнює зріст солдатів, перших у своїх колонах і вказує, якому з них треба ставати останнім у нову колону, а кому залишатися першим у своїй. Так він вчиняє, поки одна з колон не вичерпається — тоді решта іншої колони додається до нової.

   Під час сортування в дві допоміжні черги з основної поміщаються перші дві відсортовані підпослідовності, які потім зливаються в одну і результат записується в тимчасову чергу. Потім з основної черги беруться наступні дві відсортовані підпослідовності і так до тих пір доки основна черга не стане порожньою. Після цього послідовність з тимчасової черги переміщається в основну чергу. І знову продовжується сортування злиттям двох відсортованих підпослідовностей. Сортування триватиме до тих пір поки довжина відсортованої підпослідовності не стане рівною довжині самої послідовності.

   1. Аналіз алгоритму

      Час роботи алгоритму $T(n)$ по впорядкуванню елементів задовільняє рекурентному співвідношенню:

$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$, де $T\left(\frac{n}{2}\right)$ – час на впорядкування половини масиву, $O(n)$ – час на злиття цих половинок.

Враховуючи, що $T(1) = O(1)$, розв'язком співвідношення є $T(n) = O(n \log\ n)$.

Крім того, алгоритм потребує для своєї роботи $O(n)$ додаткової пам'яті.

Алгоритм не міняє порядок розташування однакових елементів, а отже він є стабільним.

3. Опис структури програми та особливостей реалізації.

   Програма має функції merge_sort, bubleSort та bubleSortFlagged, що відповідають сортуванню злиттям та сортуванням бульбашкою, сортуванням бульбашкою з флажком.

   У головній функції виконує такі дії:

   1. Якщо задано три аргументи у коммандному рядку, то використовує їх значення, то запитує ці значення для таких змінних:
      1. n – розмір першого тестового масиву для перевірки коректності сортування
      2. min - мінімальне можливе число для генерації масивів
      3. max – максимальне можливе число для генерації масивів
   2. Створює файл statistics.csv, де буде зберігатися статистика виконання функцій
   3. Створює файли src.txt і dst.txt, у src.txt зберігається масиви на три випадки:
      1. З відсортованими значеннями – для кращого випадку
      2. З випадковими числами – для середнього випадку
      3. З відсортованими значеннями у протилежному порядку – для найгіршого випадку

      У dst.txt зберігається 9 відсортованих масивів і до кожного з випадків по три функції merge_sort, bubleSort та bubleSortFlagged відповідно.
   4. Виконує цикл зі збільшенням розміру масиву з такими діями:
      1. Ініціює масиви A, B
      2. Масив A заповнюється відсортованими числами
      3. Якщо це перший тест, то виводить масив у файл src.txt
      4. Виконує сортування злиттям та вимірює час виконання

5. Звільнює пам’ять буферного масиву, так щоб основний масив був у змінній A
6. Якщо це перший тест, то виводить отриманий масив у файл dst.txt
7. Виконує сортування бульбашкою та вимірює час виконання
8. Якщо це перший тест, то виводить отриманий масив у файл dst.txt
9. Виконує сортування бульбашкою з флажком та вимірює час виконання
10. Якщо це перший тест, то виводить отриманий масив у файл dst.txt
11. Ініціює масиви B і C
12. Заповнює A випадковими числами та копіює масив у B
13. Якщо це перший тест, то виводить масив A у файл src.txt
14. Виконує сортування злиттям масиву B та вимірює час виконання
15. Звільнює пам’ять буферного масиву (B або C), так щоб основний масив був у змінній B
16. Якщо це перший тест, то виводить отриманий масив у файл dst.txt
17. Копіює масив A в B
18. Виконує сортування бульбашкою масиву B та вимірює час виконання
19. Якщо це перший тест, то виводить отриманий масив у файл dst.txt
20. Копіює масив A в B
21. Виконує сортування бульбашкою з флажком масиву B та вимірює час виконання
22. Якщо це перший тест, то виводить отриманий масив у файл dst.txt
23. Заповнює масив A відсортовані у протилежному порядку числа
24. Виконує подібні дії як у пунктах 13-22
25. Виводить статистику у statistics.csv
26. Якщо це перший тест, то зариває файли src.txt і dst.txt

27. Звільнює пам’ять масиву A

5. Закриває statistics.csv та виходить з програми

4. Тестові приклади для кожного із алгоритмів.

Вхідні масиви (Найкращий, середній, найгірший випадки відповідно):

1. 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
2. -16 -80 -68 -41 -66 35 75 -99 -89 50 -58 -99 -17 53 98 -9 33 58 79 -46 -71 76 45 62 -17 -43 41 -40 -65 70 78 -31 40 60 -22 74 -6 -48 25 -45 52 17 -44 -66 -31 -47 -25 52 -90 -46 56 -60 80 0 52 -38 8 -58 -28 43 -89 100 62 1 59 -61 -76 -97 -59 0 -42 93 -84 -35 78 -64 -32 -48 88 -71 -44 44 -81 -65 94 71 -3 1 -88 20 94 74 19 55 -76 -22 94 -52 -69 85
3. 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Вихідні масиви (До кожного з попередніх випадків три функції сортування: злиттям, бульбашкою та бульбашкою з флажком):

1. 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99

2. -99 -99 -97 -90 -89 -89 -88 -84 -81 -80 -76 -76 -71 -71 -69 -68 -66 -66 -65 -65 -64 -61 -60 -59 -58 -58 -52 -48 -48 -47 -46 -46 -45 -44 -44 -43 -42 -41 -40 -38 -35 -32 -31 -31 -28 -25 -22 -22 -17 -17 -16 -9 -6 -3 0 0 1 1 8 17 19 20 25 33 35 40 41 43 44 45 50 52 52 52 53 55 56 58 59 60 62 62 70 71 74 74 75 76 78 78 79 80 85 88 93 94 94 94 98 100
   -99 -99 -97 -90 -89 -89 -88 -84 -81 -80 -76 -76 -71 -71 -69 -68 -66 -66 -65 -65 -64 -61 -60 -59 -58 -58 -52 -48 -48 -47 -46 -46 -45 -44 -44 -43 -42 -41 -40 -38 -35 -32 -31 -31 -28 -25 -22 -22 -17 -17 -16 -9 -6 -3 0 0 1 1 8 17 19 20 25 33 35 40 41 43 44 45 50 52 52 52 53 55 56 58 59 60 62 62 70 71 74 74 75 76 78 78 79 80 85 88 93 94 94 94 98 100
   -99 -99 -97 -90 -89 -89 -88 -84 -81 -80 -76 -76 -71 -71 -69 -68 -66 -66 -65 -65 -64 -61 -60 -59 -58 -58 -52 -48 -48 -47 -46 -46 -45 -44 -44 -43 -42 -41 -40 -38 -35 -32 -31 -31 -28 -25 -22 -22 -17 -17 -16 -9 -6 -3 0 0 1 1 8 17 19 20 25 33 35 40 41 43 44 45 50 52 52 52 53 55 56 58 59 60 62 62 70 71 74 74 75 76 78 78 79 80 85 88 93 94 94 94 98 100

1. 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53

```
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

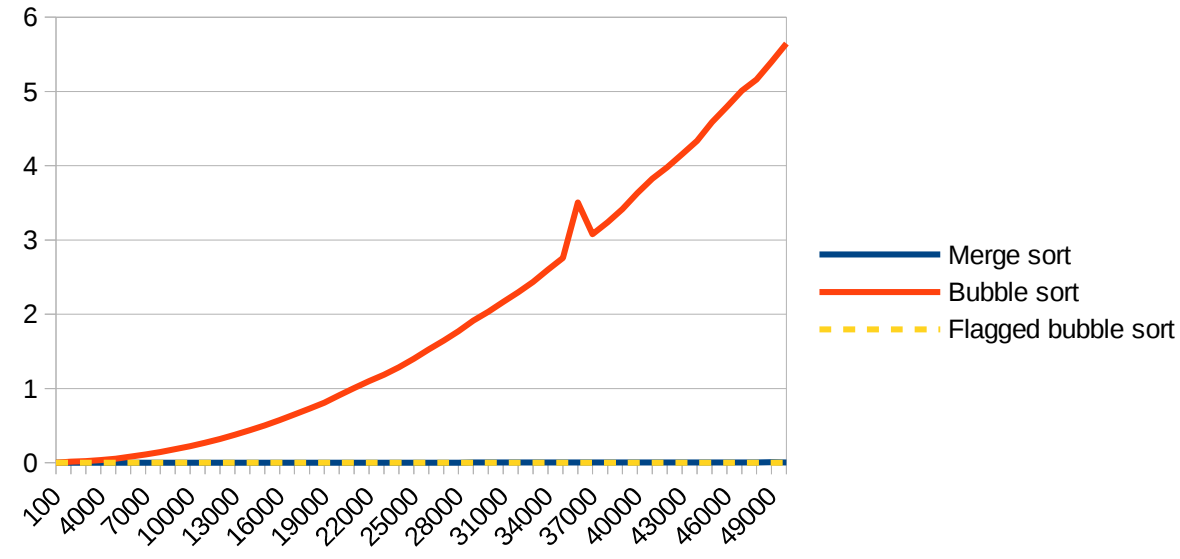
Час виконання сортуванням злиття 0.000100 секунд, а сортуванням бульбашкою 0.000590 с. , що приблизно у 6 разів повільніше.

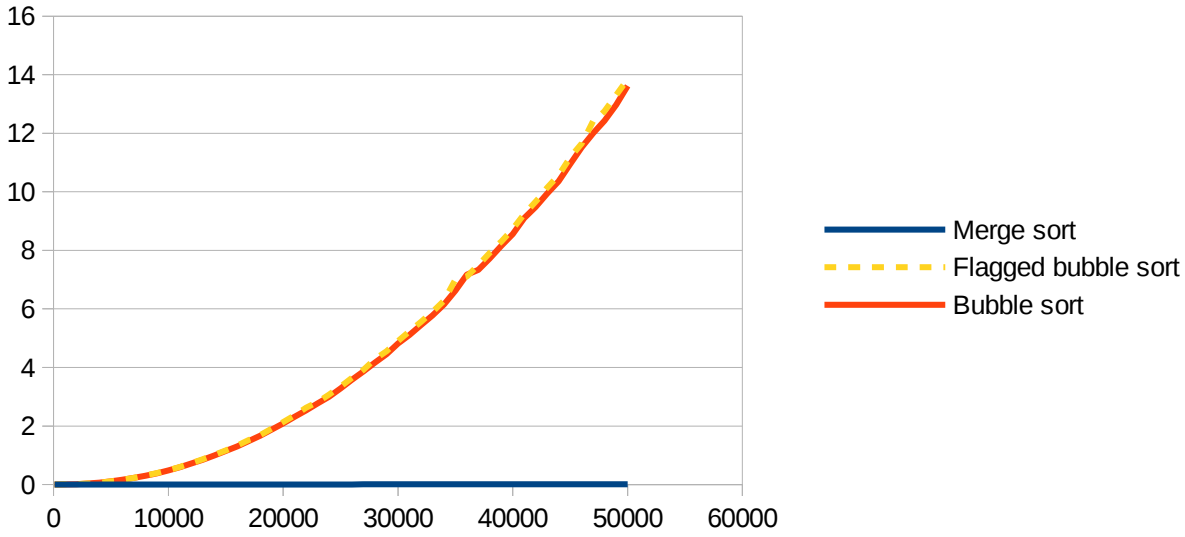
1. Для найкращого випадку часи виконання такі:
    1. 0.000021 – Сортування злиттям
    2. 0.000057 – Сортування бульбашкою
    3. 0.000003 – Сортування бульбашкою з флажком
2. Для середнього випадку:
    1. 0.000031 – Сортування злиттям
    2. 0.000106 – Сортування бульбашкою
    3. 0.000108 – Сортування бульбашкою з флажком
3. Для найгіршого випадку:
    1. 0.000018 – Сортування злиттям
    2. 0.000100 – Сортування бульбашкою
    3. 0.000102 – Сортування бульбашкою з флажком

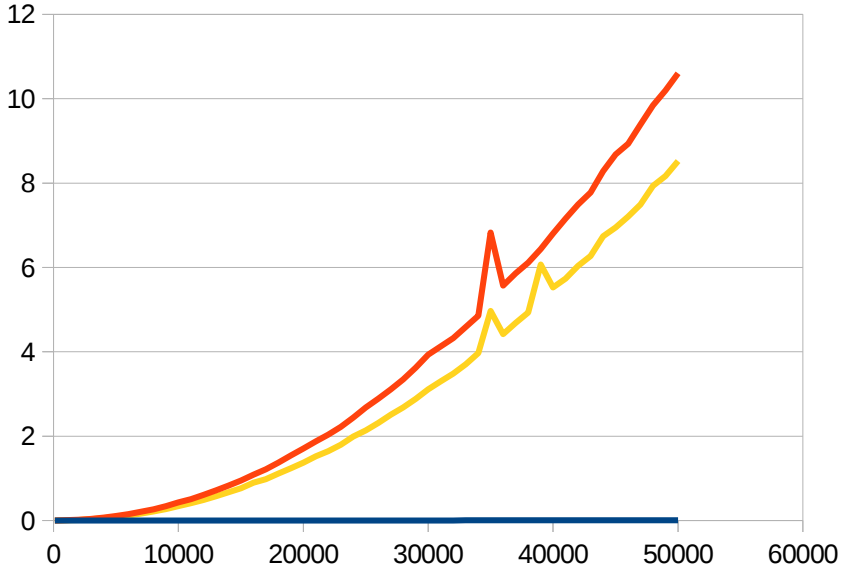
5. Порівняльний аналіз алгоритмів (таблиці, графіки)

Best time			Common time			Worst time		
Merge sort	Bubble sort	Flagged bubble sort	Merge sort	Bubble sort	Flagged bubble sort	Merge sort	Bubble sort	Flagged bubble sort
0.000021	0.000057	0.000003	0.000031	0.000106	0.000108	0.000018	0.0001	0.000102
0.000427	0.01563	0.00001	0.000351	0.014908	0.014998	0.000173	0.016751	0.013592
0.000262	0.020304	0.000015	0.000528	0.034703	0.034935	0.000268	0.037759	0.030708
0.000355	0.035779	0.000019	0.000714	0.063563	0.064291	0.000367	0.067163	0.054431
0.000458	0.056006	0.000023	0.000912	0.106244	0.107052	0.000473	0.105723	0.086836
0.000556	0.084677	0.000029	0.00123	0.161572	0.159599	0.00057	0.151613	0.122625
0.00066	0.110774	0.000034	0.001365	0.227711	0.230634	0.000678	0.206107	0.166394
0.000782	0.144023	0.000038	0.001492	0.300677	0.305487	0.000782	0.268589	0.217048
0.000883	0.182095	0.000043	0.001683	0.385578	0.391041	0.000931	0.340598	0.274724
0.000975	0.224702	0.000047	0.00196	0.487221	0.49602	0.001009	0.426628	0.339193
0.001101	0.270727	0.000052	0.002106	0.592436	0.602797	0.0011	0.50843	0.411083
0.001179	0.322223	0.000056	0.002254	0.721545	0.727909	0.001212	0.607223	0.489777
0.001302	0.378445	0.000061	0.002477	0.846601	0.864297	0.001334	0.713077	0.576046
0.001392	0.438636	0.000064	0.002659	0.992857	1.010824	0.001443	0.828301	0.671808
0.001502	0.503547	0.000069	0.002885	1.145322	1.166523	0.001546	0.949916	0.767992

0.0016	0.577444	0.000076	0.003096	1.306651	1.333172	0.001653	1.087562	0.900411
0.001721	0.649388	0.000081	0.003275	1.488106	1.518454	0.001762	1.214121	0.983046
0.001843	0.729212	0.000085	0.003493	1.665149	1.692946	0.001879	1.372696	1.11072
0.001948	0.808894	0.000089	0.003679	1.878277	1.912655	0.001994	1.541808	1.238768
0.002068	0.90967	0.000095	0.003866	2.084826	2.128715	0.002095	1.706299	1.373847
0.002183	1.004242	0.000099	0.004074	2.311683	2.367501	0.002216	1.876403	1.522303
0.002295	1.099715	0.000103	0.004306	2.53616	2.623441	0.002317	2.03763	1.645603
0.002407	1.184713	0.000107	0.004485	2.769388	2.823215	0.002441	2.22191	1.797036
0.002511	1.287959	0.00011	0.004681	2.999945	3.074567	0.002559	2.439179	1.990128
0.002624	1.401323	0.000117	0.004906	3.276674	3.342791	0.002644	2.677728	2.139734
0.002746	1.526917	0.000122	0.005137	3.578168	3.643374	0.002808	2.892	2.313282
0.002858	1.644772	0.000124	0.005317	3.863106	3.919382	0.002917	3.112359	2.509366
0.002975	1.769177	0.000131	0.005503	4.16156	4.279504	0.003028	3.343899	2.683897
0.00481	1.912695	0.000135	0.005688	4.456695	4.551687	0.003141	3.620263	2.880175
0.003226	2.031718	0.00014	0.005983	4.817999	4.906144	0.003258	3.931444	3.114337
0.00333	2.16499	0.000147	0.006176	5.106767	5.239294	0.003349	4.129335	3.302136
0.003439	2.294148	0.000146	0.00636	5.449196	5.553657	0.003472	4.324072	3.481426
0.003547	2.434747	0.000154	0.0066	5.772966	5.899141	0.003603	4.593543	3.703514
0.003775	2.598942	0.000159	0.006752	6.146273	6.266996	0.003713	4.857097	3.973974
0.003913	2.759084	0.000163	0.007007	6.618772	6.998094	0.004018	6.827837	4.969997
0.005818	3.506766	0.000165	0.007893	7.167429	7.103906	0.003959	5.573006	4.419009
0.00402	3.078014	0.000174	0.007334	7.337721	7.501199	0.004049	5.862208	4.684444
0.004153	3.240677	0.000178	0.007583	7.724112	7.914362	0.004181	6.113566	4.930073
0.004561	3.415596	0.000179	0.007808	8.152361	8.302298	0.004283	6.434502	6.07008
0.004391	3.631591	0.000188	0.007966	8.551498	8.739087	0.004716	6.805456	5.525245
0.004504	3.823459	0.000191	0.008185	9.102684	9.236264	0.004622	7.159821	5.732442
0.004656	3.977532	0.000195	0.008421	9.483344	9.667302	0.004656	7.49012	6.040259
0.00468	4.154098	0.000198	0.00869	9.937491	10.140636	0.00477	7.775626	6.272884
0.004867	4.332582	0.000207	0.008906	10.356506	10.564962	0.004878	8.283954	6.739696
0.004969	4.586988	0.000211	0.009063	10.944079	11.176934	0.004991	8.682455	6.949007
0.005108	4.794977	0.000214	0.009296	11.52592	11.654709	0.005171	8.931597	7.204966
0.005229	5.00866	0.000214	0.009713	12.003505	12.413534	0.005288	9.390633	7.491826
0.005331	5.162577	0.000223	0.009651	12.424963	12.748058	0.005401	9.845923	7.933476
0.006824	5.395789	0.000229	0.009826	12.972192	13.295049	0.005437	10.195917	8.168281
0.00555	5.644363	0.000234	0.010044	13.621124	13.845642	0.005566	10.603388	8.523527


6
5
4
3
2
1
0
100
4000
7000
10000
13000
16000
19000
22000
25000
28000
31000
34000
37000
40000
43000
46000
49000
Merge sort
Bubble sort
Flagged bubble sort


16
14
12
10
8
6
4
2
0
0
10000
20000
30000
40000
50000
60000
Merge sort
Flagged bubble sort
Bubble sort


12
10
8
6
4
2
0
0
10000
20000
30000
40000
50000
60000
Merge sort
Bubble sort
Flagged bubble sort

6. Висновки:
    1. За графіком видно, що сортування бульбашкою працює значно повільніше ніж сортування злиттям, причому для великого масиву розміром 50 000 сортування злиттям працює у 1369 разів швидше. Це відповідає тому що складність сортування злиттям має складність O(n log n), а складність сортування бульбашкою O(n^2)
    2. У таблиці ми бачимо, що середній випадок у сортувалки бульбашкою працює повільніше ніж найгірший випадок, це пов’язано з тим, що процесор кешує данні, а так як у найгіршого випадку відсортований масив, тому останній елемент переходить у кінець послідовно через усі елементи, а у випадкових числах елементи переходять випадково, тому не так добре кешуеться.
7. Код:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>

void ign_other(FILE * input) {
	char c = 0;
	while (c!='\n')
		c=fgetc(input);
}

int* merge_sort(int *up, int *down, unsigned int left, unsigned int right)
{
	if (left == right)
	{
		down[left] = up[left];
		return down;
	}

	unsigned int middle = (left + right) / 2;

	int *l_buff = merge_sort(up, down, left, middle);
	int *r_buff = merge_sort(up, down, middle + 1, right);

	int *target = l_buff == up ? down : up;

	unsigned int l_cur = left, r_cur = middle + 1;
	for (unsigned int i = left; i <= right; i++)
	{
		if (l_cur <= middle && r_cur <= right)
		{
			if (l_buff[l_cur] < r_buff[r_cur])
			{
				target[i] = l_buff[l_cur];
				l_cur++;
			}
			else
```

```
                {
                        target[i] = r_buff[r_cur];
                        r_cur++;
                }
        }
        else if (l_cur <= middle)
        {
                target[i] = l_buff[l_cur];
                l_cur++;
        }
        else
        {
                target[i] = r_buff[r_cur];
                r_cur++;
        }
    }
    return target;
}

void bubleSort(int *mas, int size) {
    for(int i = 0; i < size; i++) {
        for(int j = 0; j < size - i - 1; j++) {
            if(mas[j] > mas[j+1]) {
                int tmp = mas[j];
                mas[j] = mas[j+1];
                mas[j+1] = tmp;
            }
        }
    }
}

void bubleSortFlagged(int *mas, int size) {
    char f = 0,tmp;
    for(int i = 0; i < size; i++) {
        for(int j = 0; j < size - i - 1; j++) {
            if(mas[j] > mas[j+1]) {
                f = 1;
                tmp = mas[j];
                mas[j] = mas[j+1];
                mas[j+1] = tmp;
            }
        }
        if (f == 0) break;
    }
}

int main(int argc,char *argv[]) {
    srand(time(NULL));
    int * a, * b ,* c, * d, min = 0, max = 0, n = 0, i = 0;
    char first_run = 1;
    clock_t t1,t2;
    double dur[9];
    FILE * src, * dst, * stat;
    if (argc > 3) {
        sscanf(argv[1],"%i",&n);
        sscanf(argv[2],"%i",&min);
        sscanf(argv[3],"%i",&max);
        if (n < 1 || n > INT_MAX-1) return 1;
```

```
            if (min >= max) return 1;
      }
      while (n < 1 || n > INT_MAX-1) {
            printf("Введіть довжину масиву:\n");
            scanf("%i",&n);
            ign_other(stdin);
            if (n < 1 || n > INT_MAX-1) {
                  system("clear");
                  printf("Вы ввели неправильну довжину, дозволено тільки ціле число
від 1 до %i\n",INT_MAX-1);
            }
      }
      while (min >= max) {
            printf("Введіть інтервал дозволених значень в масиві:\n");
            scanf("%i%i",&min,&max);
            ign_other(stdin);
            if (min >= max) {
                  system("clear");
                  printf("Вы ввели неправильний інтервал, дозволено тільки цілі
числа, при чому перше число повинно бути менше ніж друге\n");
            }
      }
      src = fopen("src.txt","w");
      dst = fopen("dst.txt","w");
      stat =      fopen("statistics.csv","w");
      if (!src || !dst || !stat) {
            if (argc < 4)
                  printf("Невозможно создать файл\n");
            return 1;
      }
      for (; n <= 50000; n += 1000) {
            // ######### //
            // Best time //
            // ######### //
            a = (int*) calloc(n, sizeof(int));
            b = (int*) calloc(n, sizeof(int));
            for (i = 0; i < n; i++) {
                  a[i] = i;
                  if (first_run == 1) fprintf(src,"%i ",i);
            }
            if (first_run == 1) fprintf(src,"\n");

            // Merge sort
            t1 = clock();
            c = merge_sort(a,b,0,n-1);
            t2 = clock();
            dur[0] = 1.0 * (t2 - t1) / CLOCKS_PER_SEC;
            if (a == c) free(b);
            else {
                  free(a);
                  a=b;
            }
            if (first_run == 1) {
                  for (i = 0; i < n; i++)
                        fprintf(dst,"%i ",a[i]);
                  fprintf(dst,"\n");
            }
```

```
// Bubble sort
t1 = clock();
bubleSort(a,n);
t2 = clock();
dur[1] = 1.0 * (t2 - t1) / CLOCKS_PER_SEC;
if (first_run == 1) {
      for (i = 0; i < n; i++)
            fprintf(dst,"%i ",a[i]);
      fprintf(dst,"\n");
}

// Flagged bubble sort
t1 = clock();
bubleSortFlagged(a,n);
t2 = clock();
dur[2] = 1.0 * (t2 - t1) / CLOCKS_PER_SEC;
if (first_run == 1) {
      for (i = 0; i < n; i++)
            fprintf(dst,"%i ",a[i]);
      fprintf(dst,"\n\n");
}


// ########### //
// Common time //
// ########### //
b = (int*) calloc(n, sizeof(int));
c = (int*) calloc(n, sizeof(int));
for (i = 0; i < n; i++) {
      a[i]=rand()%(max - min + 1) + min;
      b[i]=a[i];
      if (first_run == 1) {
            fprintf(src,"%i ",a[i]);
      }
}
if (first_run == 1) {
      fprintf(src,"\n");
}

// Merge sort
t1 = clock();
d = merge_sort(b,c,0,n-1);
t2 = clock();
dur[3] = 1.0 * (t2 - t1) / CLOCKS_PER_SEC;
if (b == d) free(c);
else {
      free(b); b = c;
}
if (first_run == 1) {
      for (i = 0; i < n; i++)
            fprintf(dst,"%i ",b[i]);
      fprintf(dst,"\n");
}

// Bubble sort
for (i = 0; i < n; i++)
      b[i] = a[i];
t1 = clock();
```

```
bubleSort(b,n);
t2 = clock();
dur[4] = 1.0 * (t2 - t1) / CLOCKS_PER_SEC;
if (first_run == 1) {
      for (i = 0; i < n; i++)
            fprintf(dst,"%i ",b[i]);
      fprintf(dst,"\n");
}

// Flagged bubble sort
for (i = 0; i < n; i++)
      b[i] = a[i];
t1 = clock();
bubleSortFlagged(b,n);
t2 = clock();
dur[5] = 1.0 * (t2 - t1) / CLOCKS_PER_SEC;
if (first_run == 1) {
      for (i = 0; i < n; i++)
            fprintf(dst,"%i ",b[i]);
      fprintf(dst,"\n\n");
}


// ########### //
// Worst time //
// ########### //
for (i = 0; i < n; i++) {
      a[i] = n - i;
      if (first_run == 1) fprintf(src,"%i ",a[i]);
}
if (first_run == 1) fprintf(src,"\n");

// Merge sort
t1 = clock();
c = merge_sort(a,b,0,n-1);
t2 = clock();
dur[6] = 1.0 * (t2 - t1) / CLOCKS_PER_SEC;
if (a == c) free(b);
else {
      free(a);
      a=b;
}
if (first_run == 1) {
      for (i = 0; i < n; i++)
            fprintf(dst,"%i ",a[i]);
      fprintf(dst,"\n");
}

// Bubble sort
for (i = 0; i < n; i++)
      a[i] = n - i;
t1 = clock();
bubleSort(a,n);
t2 = clock();
dur[7] = 1.0 * (t2 - t1) / CLOCKS_PER_SEC;
if (first_run == 1) {
      for (i = 0; i < n; i++)
            fprintf(dst,"%i ",a[i]);
```

```
                  fprintf(dst,"\n");
            }

            // Flagged bubble sort
            for (i = 0; i < n; i++)
                  a[i] = n - i;
            t1 = clock();
            bubleSortFlagged(a,n);
            t2 = clock();
            dur[8] = 1.0 * (t2 - t1) / CLOCKS_PER_SEC;
            if (first_run == 1) {
                  for (i = 0; i < n; i++)
                        fprintf(dst,"%i ",a[i]);
                  fprintf(dst,"\n\n");
            }


            fprintf(stat,"%i,%lf,%lf,%lf,%lf,%lf,%lf,%lf,%lf,%lf\n", n, dur[0],
dur[1], dur[2], dur[3], dur[4], dur[5], dur[6], dur[7], dur[8]);
            printf("%i,%lf,%lf,%lf,%lf,%lf,%lf,%lf,%lf,%lf\n", n, dur[0], dur[1],
dur[2], dur[3], dur[4], dur[5], dur[6], dur[7], dur[8]);
            if (first_run == 1) {
                  fclose(src);
                  fclose(dst);
            }
            free(a);
            if (first_run == 1) {
                  first_run = 0;
                  n = 1000;
            }
      }
      fclose(stat);
}
```