**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

**ЛАБОРАТОРНАЯ РАБОТА №2**

**по дисциплине «Алгоритмы и структуры данных»**

| Студенты гр. 2302 |  | Николаев В.Ю. |
| --- | --- | --- |
| Преподаватель |  | Пестерев Д.О. |

Санкт-Петербург

2023

# 1. Постановка задачи

1. Реализовать следующие алгоритмы сортировки:

* Сортировка выбором
* Сортировка вставками
* Сортировка пузырьком
* Сортировка слиянием
* Быстрая сортировка
* Сортировка Шелла (не менее трех различных последовательностей, желательно приводящих к разным асимптотикам)
* Пирамидальная сортировка
* Timsort
* IntroSort

1. Экспериментально определить время работы алгоритмов при различных размерах массива для:

* Отсортированного массива
* Почти отсортированного массива
* Обратно отсортированного массива
* Массива, о структуре которого дополнительных данных не дано

Постараться определить асимптотику для лучшего/среднего/худшего случая путем анализа экспериментальных данных и применения к ним нелинейной регрессии.

1. Постараться получить достоверные данные о времени работы алгоритмов сортировки при малом размере массива ~ 100 элементов

# 2. Время работы алгоритмов

## SelectionSort

| Отсортированный массив | Почти отсортированный массив |
| --- | --- |
| Обратно отсортированного массива | Не отсортированный массив |

| Лучший | Средний | Худший |
| --- | --- | --- |
|  |  |  |

## InsertionSort

| Отсортированный массив | Почти отсортированный массив |
| --- | --- |
| Обратно отсортированного массива | Не отсортированный массив |

| Лучший | Средний | Худший |
| --- | --- | --- |
|  |  |  |

## BubbleSort

| Отсортированный массив | Почти отсортированный массив |
| --- | --- |
| Обратно отсортированного массива | Не отсортированный массив |

| Лучший | Средний | Худший |
| --- | --- | --- |
|  |  |  |

## MergeSort

| Отсортированный массив | Почти отсортированный массив |
| --- | --- |
| Обратно отсортированного массива | Не отсортированный массив |

| Лучший | Средний | Худший |
| --- | --- | --- |
|  |  |  |

## QuickSort

| Отсортированный массив | Почти отсортированный массив |
| --- | --- |
| Обратно отсортированного массива | Не отсортированный массив |

| Лучший | Средний | Худший |
| --- | --- | --- |
|  |  |  |

## ShellSort

| Отсортированный массив | Почти отсортированный массив |
| --- | --- |
| Обратно отсортированного массива | Не отсортированный массив |

| Лучший | Средний | Худший |
| --- | --- | --- |
|  |  |  |

## MarcinaCiuraShellSort

| Отсортированный массив | Почти отсортированный массив |
| --- | --- |
| Обратно отсортированного массива | Не отсортированный массив |

| Лучший | Средний | Худший |
| --- | --- | --- |
|  |  |  |

## FibonacciShellSort

| Отсортированный массив | Почти отсортированный массив |
| --- | --- |
| Обратно отсортированного массива | Не отсортированный массив |

| Лучший | Средний | Худший |
| --- | --- | --- |
|  |  |  |

## HeapSort

| Отсортированный массив | Почти отсортированный массив |
| --- | --- |
| Обратно отсортированного массива | Не отсортированный массив |

| Лучший | Средний | Худший |
| --- | --- | --- |
|  |  |  |

## TimSort

| Отсортированный массив | Почти отсортированный массив |
| --- | --- |
| Обратно отсортированного массива | Не отсортированный массив |

| Лучший | Средний | Худший |
| --- | --- | --- |
|  |  |  |

## IntroSort

| Отсортированный массив | Почти отсортированный массив |
| --- | --- |
| Обратно отсортированного массива | Не отсортированный массив |

| Лучший | Средний | Худший |
| --- | --- | --- |
|  |  |  |

# 3. Время работы на массивах с малым размером

| SelectionSort  1.2e-5 | | InsertionSort  2.0e-5 | | BubbleSort  3.0e-5 | |
| --- | --- | --- | --- | --- | --- |
| MergeSort  4.2e-6 | | QuickSort  2.2e-6 | | HeapSort  8.3e-6 | |
| MarcinaCiuraShellSort  2.1e-6 | | FibonacciShellSort  3.1e-6 | | ShellSort  2.4e-6 | |
| TimSort  1.1e-5 | | | IntroSort  2.2e-6 | | |

Самой быстрой оказалась MarcinaCiuraShellSort, однако разница между ней и сортировками QuickSort и IntroSort всего 1e-7.

# 4. Пример работы

|  |
| --- |

# 5. Листинг

| | lib/BubbleSort.h #pragma once  template <typename type\_arr>  void BubbleSort(type\_arr \*arr, int size)  {  for (int i = 0; i < size - 1; i++)  for (int j = size - 1; j > i; j--)  if (arr[j] < arr[j - 1])  std::swap(arr[j], arr[j - 1]);  } | | --- |  | lib/HeapSort.h #pragma once  template <typename type\_arr>  void Heapify(type\_arr \*arr, int size, int i)  {  int largest = i;  int left = 2 \* i + 1;  int right = 2 \* i + 2;  if (left < size && arr[left] > arr[largest])  largest = left;  if (right < size && arr[right] > arr[largest])  largest = right;  if (largest != i)  {  std::swap(arr[i], arr[largest]);  Heapify(arr, size, largest);  }  }  template <typename type\_arr>  void HeapSort(type\_arr \*arr, int size)  {  for (int i = size / 2 - 1; i >= 0; i--)  Heapify(arr, size, i);  for (int i = size - 1; i >= 0; i--)  {  std::swap(arr[0], arr[i]);  Heapify(arr, i, 0);  }  } | | --- |  | lib/InsertionSort.h #pragma once  template <typename type\_arr>  void InsertionSort(type\_arr \*arr, int size)  {  for (int i = 1; i < size; i++)  {  for (int j = i;j > 0 && arr[j] < arr[j - 1]; j--)  std::swap(arr[j], arr[j - 1]);  }  } | | --- |  | lib/IntroSort.h #pragma once  template <typename type\_arr>  void IntroSort(type\_arr \*arr, int size)  {  if (size <= 1)  return;  int pivot = arr[size - 1];  int i = 0, j = size - 2;  while (i <= j)  {  while (arr[i] < pivot)  i++;  while (arr[j] > pivot)  j--;  if (i <= j)  std::swap(arr[i++], arr[j--]);  }  std::swap(arr[i], arr[size - 1]);  IntroSort(arr, i);  IntroSort(arr + i + 1, size - i - 1);  } | | --- |  | lib/MergeSort.h #pragma once  template <typename type\_arr>  void MergeSort(type\_arr \*arr, int size)  {  if (size <= 1)  return;  int mid = size / 2;  MergeSort(arr, mid);  MergeSort(arr + mid, size - mid);  type\_arr \*temp = new type\_arr[size];  int i = 0, j = mid, k = 0;  while (i < mid && j < size)  {  if (arr[i] < arr[j])  temp[k++] = arr[i++];  else  temp[k++] = arr[j++];  }  while (i < mid)  temp[k++] = arr[i++];  while (j < size)  temp[k++] = arr[j++];  for (int i = 0; i < size; i++)  arr[i] = temp[i];  delete[] temp;  } | | --- |  | lib/QuickSort.h #pragma once  template <typename type\_arr>  void QuickSort(type\_arr \*arr, int size)  {  if (size <= 1)  return;  int pivot = arr[size - 1];  int i = 0, j = size - 2;  while (i <= j)  {  while (arr[i] < pivot)  i++;  while (arr[j] > pivot)  j--;  if (i <= j)  std::swap(arr[i++], arr[j--]);  }  std::swap(arr[i], arr[size - 1]);  QuickSort(arr, i);  QuickSort(arr + i + 1, size - i - 1);  } | | --- |  | lib/SelectionSort.h #pragma once  template <typename type\_arr>  void SelectionSort(type\_arr \*arr, int size)  {  for (int i = 0; i < size - 1; i++)  {  int min = i;  for (int j = i + 1; j < size; j++)  if (arr[j] < arr[min])  min = j;  std::swap(arr[i], arr[min]);  }  } | | --- |  | lib/ShellSort.h #pragma once  template <typename type\_arr>  void GapShellSort(type\_arr \*arr, int size, int \*gaps, int gapSize)  {  for (int i = 0; i < gapSize; i++)  {  int gap = gaps[i];  for (int j = gap; j < size; j++)  {  type\_arr temp = arr[j];  int k = j;  while (k >= gap && arr[k - gap] > temp)  {  arr[k] = arr[k - gap];  k -= gap;  }  arr[k] = temp;  }  }  }  template <typename type\_arr>  void ShellSort(type\_arr \*arr, int size)  {  int len = 0;  for (int gap = size / 2; gap > 0; gap /= 2)  len++;  int \*gaps = new int[len];  int i = 0;  for (int gap = size / 2; gap > 0; gap /= 2)  gaps[i++] = gap;  GapShellSort(arr, size, gaps, len);  delete[] gaps;  }  template <typename type\_arr>  void MarcinaCiuraShellSort(type\_arr \*arr, int size)  {  int len = 11;  int gaps[] = {8858, 3937, 1750, 701, 301, 132, 57, 23, 10, 4, 1};  GapShellSort(arr, size, gaps, len);  }  template <typename type\_arr>  void FibonacciShellSort(type\_arr \*arr, int size)  {  int len = 17;  int gaps[] = {2584, 1597, 987, 610, 377, 233, 144, 89, 55, 34, 21, 13, 8, 5, 3, 2, 1};  GapShellSort(arr, size, gaps, len);  } | | --- |  | lib/Sorts.h #pragma once  #include <cstdlib>  #include <iostream>  #include <chrono>  #include <ctime>  #include <fstream>  #include "../lib/InsertionSort.h"  #include "../lib/SelectionSort.h"  #include "../lib/BubbleSort.h"  #include "../lib/MergeSort.h"  #include "../lib/QuickSort.h"  #include "../lib/ShellSort.h"  #include "../lib/HeapSort.h"  #include "../lib/TimSort.h"  #include "../lib/IntroSort.h"  template <typename type\_arr>  void Sort(type\_arr \*arr, int size, int sortNum)  {  switch (sortNum)  {  case -1:  break;  case 0:  SelectionSort(arr, size);  break;  case 1:  InsertionSort(arr, size);  break;    case 2:  BubbleSort(arr, size);  break;    case 3:  MergeSort(arr, size);  break;  case 4:  QuickSort(arr, size);  break;  case 5:  ShellSort(arr, size);  break;  case 6:  MarcinaCiuraShellSort(arr, size);  break;  case 7:  FibonacciShellSort(arr, size);  break;  case 8:  HeapSort(arr, size);  break;  case 9:  TimSort(arr, size);  break;  case 10:  IntroSort(arr, size);  break;  }  } | | --- |  | lib/TimSort.h #pragma once  #include <iostream>  #include "InsertionSort.h"  template <typename type\_arr>  void Merge(type\_arr \*arr, int left, int mid, int right)  {  int sizeLeft = mid - left + 1;  int sizeRight = right - mid;  type\_arr \*tempLeft = new type\_arr[sizeLeft];  type\_arr \*tempRight = new type\_arr[sizeRight];  for (int i = 0; i < sizeLeft; i++)  tempLeft[i] = arr[left + i];  for (int i = 0; i < sizeRight; i++)  tempRight[i] = arr[mid + 1 + i];  int i = 0, j = 0, k = left;  while (i < sizeLeft && j < sizeRight)  {  if (tempLeft[i] <= tempRight[j])  arr[k++] = tempLeft[i++];  else  arr[k++] = tempRight[j++];  }  while (i < sizeLeft)  arr[k++] = tempLeft[i++];  while (j < sizeRight)  arr[k++] = tempRight[j++];  delete[] tempLeft;  delete[] tempRight;  }  int GetMinRun(int n)  {  int r = 0;  while (n >= 64) {  r |= n & 1;  n >>= 1;  }  return n + r;  }  template <typename type\_arr>  void TimSort(type\_arr \*arr, int size)  {  if (size <= 1)  return;  int minRun = GetMinRun(size);  for (int i = 0; i < size; i += minRun)  InsertionSort(arr + i, std::min(minRun, size - i));  for (int sizeRun = minRun; sizeRun < size; sizeRun \*= 2)  {  for (int i = 0; i < size; i += 2 \* sizeRun)  {  int mid = std::min(i + sizeRun - 1, size - 1);  int right = std::min(i + 2 \* sizeRun - 1, size - 1);  Merge(arr, i, mid, right);  }  }  } | | --- |  | src/Example.cpp #include "../lib/Sorts.h"  template <typename type\_arr>  void PrintArray(type\_arr \*arr, int size, int sortNum)  {  static const char \*sortName[] =  {  "Before sort: ",  "After SelectionSort: ",  "After InsertionSort: ",  "After BubbleSort: ",  "After MergeSort: ",  "After QuickSort: ",  "After ShellSort: ",  "After MarcinaCiuraShellSort: ",  "After FibonacciShellSort: ",  "After HeapSort: ",  "After TimSort: ",  "After IntroSort: "  };  std::cout << sortName[sortNum];  for (int i = 0; i < size; i++)  std::cout << arr[i] << " ";  std::cout << std::endl;  }  int main()  {  int size = 9;  int \*arr = new int[size];  for (int i = 0; i < size; i++)  arr[i] = rand() % 100;  int \*coppyArr = new int[size];  int sortNums = 11;  for (int i = -1; i < sortNums; i++)  {  for (int j = 0; j < size; j++)  coppyArr[j] = arr[j];  Sort(coppyArr, size, i);  PrintArray(coppyArr, size, i);  }  delete[] arr;  delete[] coppyArr;  return 0;  } | | --- |  | src/Plot.py import os  import matplotlib.pyplot as plt  ss = [  "Checking",  "CheckingSorted",  "CheckingAlmostSorted",  "CheckingReverseSorted",  "CheckingOnSmall"  ]  for s in ss:  path = "results/" + s  files = os.listdir(path)  for file in files:  f = open(path + "/" + file, "r")  x, y = [[float(i) for i in line.split()] for line in f.read().splitlines()]  plt.plot(x, y, "b.")  # p1, p2 = [x[0], x[-1]], [y[0], y[-1]]  # plt.plot(p1, p2, "r-")  plt.xlabel("array size")  plt.ylabel("sorting time (s)")  plt.title(file[:-4])  plt.savefig("png/" + s + "/" + file[:-4] + ".png")  # plt.show()  plt.clf() | | --- | | | src/Checking.cpp #include "../lib/Sorts.h"  int main()  {  const char \*sortName[] =  {  "results/Checking/SelectionSort.txt",  "results/Checking/InsertionSort.txt",  "results/Checking/BubbleSort.txt",  "results/Checking/MergeSort.txt",  "results/Checking/QuickSort.txt",  "results/Checking/ShellSort.txt",  "results/Checking/MarcinaCiuraShellSort.txt",  "results/Checking/FibonacciShellSort.txt",  "results/Checking/HeapSort.txt",  "results/Checking/TimSort.txt",  "results/Checking/IntroSort.txt"  };  int size = 50000;  int \*arr = new int[size];    for (int i = 0; i < size; i++)  arr[i] = rand() % 10000;  int points = 100;  int sortNums = 11;  for (int sortNum = 0; sortNum < sortNums; sortNum++)  {  std::fstream file(sortName[sortNum], std::ios::out);  for (int i = 0; i < points; i++)  {  int copySize = size / points \* (i + 1);  file << copySize << ' ';  }  file << std::endl;  for (int i = 0; i < points; i++)  {  int copySize = size / points \* (i + 1);  int \*copyArr = new int[copySize];  for (int j = 0; j < copySize; j++)  copyArr[j] = arr[j];  auto start = std::chrono::system\_clock::now();  Sort(copyArr, copySize, sortNum);  auto end = std::chrono::system\_clock::now();  std::chrono::duration<double> elapsed\_seconds = end - start;  file << elapsed\_seconds.count() << ' ';  delete[] copyArr;  }  }  delete[] arr;  return 0;  } | | --- |  | src/CheckingAlmostSorted.cpp #include "../lib/Sorts.h"  int main()  {  const char \*sortName[] =  {  "results/CheckingAlmostSorted/SelectionSort.txt",  "results/CheckingAlmostSorted/InsertionSort.txt",  "results/CheckingAlmostSorted/BubbleSort.txt",  "results/CheckingAlmostSorted/MergeSort.txt",  "results/CheckingAlmostSorted/QuickSort.txt",  "results/CheckingAlmostSorted/ShellSort.txt",  "results/CheckingAlmostSorted/MarcinaCiuraShellSort.txt",  "results/CheckingAlmostSorted/FibonacciShellSort.txt",  "results/CheckingAlmostSorted/HeapSort.txt",  "results/CheckingAlmostSorted/TimSort.txt",  "results/CheckingAlmostSorted/IntroSort.txt"  };  int size = 50000;  int \*arr = new int[size];    for (int i = 0; i < size; i++)  arr[i] = rand() % 10000;  QuickSort(arr, size);  int points = 100;  for (int i = 0; i < points; i++)  {  int copySize = size / points \* (i + 1);  std::swap(arr[copySize - 2], arr[copySize - 1]);  }  int sortNums = 11;  for (int sortNum = 0; sortNum < sortNums; sortNum++)  {  std::fstream file(sortName[sortNum], std::ios::out);  for (int i = 0; i < points; i++)  {  int copySize = size / points \* (i + 1);  file << copySize << ' ';  }  file << std::endl;  for (int i = 0; i < points; i++)  {  int copySize = size / points \* (i + 1);  int \*copyArr = new int[copySize];  for (int j = 0; j < copySize; j++)  copyArr[j] = arr[j];  auto start = std::chrono::system\_clock::now();  Sort(copyArr, copySize, sortNum);  auto end = std::chrono::system\_clock::now();  std::chrono::duration<double> elapsed\_seconds = end - start;  file << elapsed\_seconds.count() << ' ';  delete[] copyArr;  }  }  delete[] arr;  return 0;  } | | --- |  | src/CheckingOnSmall.cpp #include "../lib/Sorts.h"  int main()  {  const char \*sortName[] =  {  "results/CheckingOnSmall/SelectionSort.txt",  "results/CheckingOnSmall/InsertionSort.txt",  "results/CheckingOnSmall/BubbleSort.txt",  "results/CheckingOnSmall/MergeSort.txt",  "results/CheckingOnSmall/QuickSort.txt",  "results/CheckingOnSmall/ShellSort.txt",  "results/CheckingOnSmall/MarcinaCiuraShellSort.txt",  "results/CheckingOnSmall/FibonacciShellSort.txt",  "results/CheckingOnSmall/HeapSort.txt",  "results/CheckingOnSmall/TimSort.txt",  "results/CheckingOnSmall/IntroSort.txt"  };  int sortNums = 11;  int size = 100;  int \*arr = new int[size];    for (int i = 0; i < size; i++)  arr[i] = rand() % 10000;  int points = 100;  for (int sortNum = 0; sortNum < sortNums; sortNum++)  {  std::fstream file(sortName[sortNum], std::ios::out);    for (int i = 0; i < points; i++)  file << i << ' ';  file << std::endl;  for (int i = 0; i < points; i++)  {  int copySize = size;  int \*copyArr = new int[copySize];  for (int j = 0; j < copySize; j++)  copyArr[j] = arr[j];  auto start = std::chrono::system\_clock::now();  Sort(copyArr, copySize, sortNum);  auto end = std::chrono::system\_clock::now();  std::chrono::duration<double> elapsed\_seconds = end - start;  file << elapsed\_seconds.count() << ' ';  delete[] copyArr;  }  }  delete[] arr;  return 0;  } | | --- |  | src/CheckingReverseSorted.cpp #include "../lib/Sorts.h"  int main()  {  const char \*sortName[] =  {  "results/CheckingReverseSorted/SelectionSort.txt",  "results/CheckingReverseSorted/InsertionSort.txt",  "results/CheckingReverseSorted/BubbleSort.txt",  "results/CheckingReverseSorted/MergeSort.txt",  "results/CheckingReverseSorted/QuickSort.txt",  "results/CheckingReverseSorted/ShellSort.txt",  "results/CheckingReverseSorted/MarcinaCiuraShellSort.txt",  "results/CheckingReverseSorted/FibonacciShellSort.txt",  "results/CheckingReverseSorted/HeapSort.txt",  "results/CheckingReverseSorted/TimSort.txt",  "results/CheckingReverseSorted/IntroSort.txt"  };  int size = 50000;  int \*arr = new int[size];    for (int i = 0; i < size; i++)  arr[i] = rand() % 10000;  QuickSort(arr, size);  for (int i = 0; i < size / 2; i++)  std::swap(arr[i], arr[size - i - 1]);  int points = 100;  int sortNums = 11;  for (int sortNum = 0; sortNum < sortNums; sortNum++)  {  std::fstream file(sortName[sortNum], std::ios::out);  for (int i = 0; i < points; i++)  {  int copySize = size / points \* (i + 1);  file << copySize << ' ';  }  file << std::endl;  for (int i = 0; i < points; i++)  {  int copySize = size / points \* (i + 1);  int \*copyArr = new int[copySize];  for (int j = 0; j < copySize; j++)  copyArr[j] = arr[j];  auto start = std::chrono::system\_clock::now();  Sort(copyArr, copySize, sortNum);  auto end = std::chrono::system\_clock::now();  std::chrono::duration<double> elapsed\_seconds = end - start;  file << elapsed\_seconds.count() << ' ';  delete[] copyArr;  }  }  delete[] arr;  return 0;  } | | --- |  | src/CheckingSorted.cpp #include "../lib/Sorts.h"  int main()  {  const char \*sortName[] =  {  "results/CheckingSorted/SelectionSort.txt",  "results/CheckingSorted/InsertionSort.txt",  "results/CheckingSorted/BubbleSort.txt",  "results/CheckingSorted/MergeSort.txt",  "results/CheckingSorted/QuickSort.txt",  "results/CheckingSorted/ShellSort.txt",  "results/CheckingSorted/MarcinaCiuraShellSort.txt",  "results/CheckingSorted/FibonacciShellSort.txt",  "results/CheckingSorted/HeapSort.txt",  "results/CheckingSorted/TimSort.txt",  "results/CheckingSorted/IntroSort.txt"  };  int size = 50000;  int \*arr = new int[size];    for (int i = 0; i < size; i++)  arr[i] = rand() % 10000;  QuickSort(arr, size);  int points = 100;  int sortNums = 11;  for (int sortNum = 0; sortNum < sortNums; sortNum++)  {  std::fstream file(sortName[sortNum], std::ios::out);  for (int i = 0; i < points; i++)  {  int copySize = size / points \* (i + 1);  file << copySize << ' ';  }  file << std::endl;  for (int i = 0; i < points; i++)  {  int copySize = size / points \* (i + 1);  int \*copyArr = new int[copySize];  for (int j = 0; j < copySize; j++)  copyArr[j] = arr[j];  auto start = std::chrono::system\_clock::now();  Sort(copyArr, copySize, sortNum);  auto end = std::chrono::system\_clock::now();  std::chrono::duration<double> elapsed\_seconds = end - start;  file << elapsed\_seconds.count() << ' ';  delete[] copyArr;  }  }  delete[] arr;  return 0;  } | | --- |  | src/DeleteKesh.py import os  directory\_paths = [  "results",  "png"  ]  # loop through all the files in the directories and delete them  for directory\_path in directory\_paths:  folders = os.listdir(directory\_path)  for folder in folders:  files = os.listdir(directory\_path + "/" + folder)  for file in files:  os.remove(directory\_path + "/" + folder + "/" + file) | | --- | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |