Лабораторная работа № 1

Роль алгоритмов в вычислениях

Теоретические задания

1. Какими еще параметрами, кроме скорости, можно характеризовать алгоритм на практике?

2. Что общего между задачей об определении кратчайшего пути и задачей о коммивояжере? Чем они различаются?

3. Предположим, на одной и той же машине проводится сравнительный анализ реализаций двух алгоритмов сортировки, работающих по методу вставок и по методу слияния. Для сортировки n элементов методом вставок необходимо $8\,n^2$ шагов, а для сортировки методом слияния – $64\,n\,lg\,n$ шагов. При каком значении п время сортировки методом вставок превысит время сортировки методом слияния?

4. При каком минимальном значении n алгоритм, время работы которого определяется формулой $100\,n^2$, работает быстрее, чем алгоритм, время работы которого выражается как 2^n, если оба алгоритма выполняются на одной и той же машине?

Практические задания

1. Сравнение времени работы алгоритмов.

Ниже приведена таблица, строки которой соответствуют различным функциям $f(n)$, а столбцы – значениям времени t. Определите максимальные значения n, для которых задача может быть решена за время t, если предполагается, что время работы алгоритма, необходимое для решения задачи, равно $f(n)$ микросекунд.

	1 секунда	1 минута	1 час	1 день	1 месяц	1 год	1 век
$lg\ n$							
$\sqrt{n}$							
n							
$n\ lg\ n$							
n^2							
n^3							
2^n							
$n!$							

Лабораторная работа № 2

Интуитивные алгоритмы сортировки

Теоретические сведения

Задача сортировки (sorting problem) формально определяется следующим образом.

Вход: последовательность из n чисел $\langle a_1, a_2, \ldots, a_n \rangle$.

Выход: перестановка (изменение порядка) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ входной последовательности таким образом, что для ее членов выполняется соотношение $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Пример

Вход: $\langle 31, 41, 59, 26, 41, 58 \rangle$

Выход: $\langle 26, 31, 41, 41, 58, 59 \rangle$

Сортировка вставкой (insertion sort)

Идея

Каждый a_j элемент последовательности, начиная со второго, вставить в подходящее место в ранее отсортированную совокупность $a_1, a_2, \ldots, a_{j-1}$. Это место определяется последовательным сравнением элемента a_j с упорядоченными элементами $a_1, a_2, \ldots, a_{j-1}$.

Пример

Исходный массив: $A = \langle 5, 2, 4, 6, 1, 3 \rangle$

После вставки a_2 элемента: $A = \langle \underline{2, 5}, 4, 6, 1, 3 \rangle$

После вставки a_3 элемента: $A = \langle \underline{2, 4, 5}, 6, 1, 3 \rangle$

После вставки a_4 элемента: $A = \langle \underline{2, 4, 5, 6}, 1, 3 \rangle$

После вставки a_5 элемента: $A = \langle \underline{1, 2, 4, 5, 6}, 3 \rangle$

После вставки a_6 элемента: $A = \langle \underline{1, 2, 3, 4, 5, 6} \rangle$

Алгоритм

INSERTION_SORT (*A*)

```
1   for j = 2 to A.length
2       key = A[j]
3       // Insert A[j] into the sorted sequence A[1 .. j – 1].
4       i = j – 1
5       while i > 0 and A[i] > key
6           A[i + 1] = A[i]
7           i = i – 1
8       A[i + 1] = key
```

Характеристики

- Время работы алгоритма: $\Theta(n^2)$
- Не использует дополнительную память
- Алгоритм устойчив

Реализация

При написании функции на языке C, принимающей в качестве аргумента массив, необходимо передать не только адрес начала данного массива в оперативной памяти (`int A[]` или `int *A`), но и количество элементов в этом массиве (`int A_length`).

```
void Insertion_Sort(int A[], int A_length) {
   int i, j, key;

   for (j=2; j <= A_length; j++) {
      key = A[j];
      // Insert A[j] into the sorted sequence A[1..j-1]
      i = j - 1;
      while (i > 0 && A[i] > key) {
         A[i+1] = A[i];
         i--;
      }
      A[i+1] = key;
   }
}
```

Как известно, элементы массива в языке C индексируются с `0`. Например, при объявлении `int X[N]` нам доступны `X[0]`, `X[1]`, …, `X[N-1]` элементы. Для того, чтобы наша программа на языке C была наиболее близка к псевдокоду, т.е. требовала минимальной адаптации, удобно объявить сортируемый массив длиной не `N`, а `N+1`. В этом случае мы будем использовать часть массива – элементы с `1` по `N`-й.

Для работы будет полезна также функция `Print_Array`, выводящая на экран элементы массива.

```
void Print_Array(int A[], int A_length) {
   int i;

   for (i=1; i <= A_length; i++)
      printf("%d ", A[i]);
   printf("\n");
}
```

В этом случае функция `main` может содержать объявление, инициализацию массива и вызовы функций `Print_Array` (до и после сортировки) и собственно `Insertion_Sort`.

```
int main(void) {
   int X[N+1]   = {0, 5, 2, 4, 6, 1, 3},
       X_length = N;

   Print_Array(X, X_length);
   Insertion_Sort(X, X_length);
   Print_Array(X, X_length);

   return 0;
}
```

Сортировка выбором (selection sort)

Идея

Сначала определяется наименьший элемент массива A, который ставится на место элемента $A[1]$, затем производится поиск второго наименьшего элемента массива A, который ставится на место элемента $A[2]$. Этот процесс продолжается для первых $n - 1$ элементов массива A.

Пример

Исходный массив: $A = \langle 5, 2, 4, 6, 1, 3 \rangle$

После обмена с a_1 элементом: $A = \langle \underline{1}, 2, 4, 6, 5, 3 \rangle$

После обмена с a_2 элементом: $A = \langle \underline{1, 2}, 4, 6, 5, 3 \rangle$

После обмена с a_3 элементом: $A = \langle \underline{1, 2, 3}, 6, 5, 4 \rangle$

После обмена с a_4 элементом: $A = \langle \underline{1, 2, 3, 4}, 5, 6 \rangle$

После обмена с a_5 элементом: $A = \langle \underline{1, 2, 3, 4, 5}, 6 \rangle$

Алгоритм

SELECTION_SORT (A)

```
 1  for j = 1 to A.length – 1
 2        min = A[j]
 3        imin = j
 4        for i = j + 1 to A.length
 5              if A[i] < min
 6                    min = A[i]
 7                    imin = i
 8        A[imin] = A[j]
```

9 $A[j] = min$

Характеристики

- Время работы алгоритма: $\Theta(n^2)$
- Не использует дополнительную память
- Алгоритм устойчив

Реализация

```
void Selection_Sort(int A[], int A_length) {
   int i, j, min, imin;

   for (j=1; j <= A_length-1; j++) {
      min  = A[j];
      imin = j;
      for (i=j+1; i <= A_length; i++)
         if (A[i] < min) {
            min  = A[i];
            imin = i;
         }
      A[imin] = A[j];
      A[j]    = min;
   }
}
```

Теоретические задания

1. Проиллюстрируйте работу алгоритмов INSERTION_SORT и SELECTION_SORT по упорядочению массива: $A = \langle 31, 41, 59, 26, 41, 58 \rangle$.

2. Перепишите процедуры INSERTION_SORT и SELECTION_SORT так, чтобы они выполняла сортировку не в невозрастающем, а в неубывающем порядке.

3. Рассмотрим задачу поиска.

Вход: последовательность n чисел $A = \langle a_1, a_2, \ldots, a_n \rangle$ и величина v.
Выход: индекс i, обладающий свойством $v = A[i]$ или значение NIL, если в массиве A отсутствует значение v.
Составьте псевдокод линейного поиска, при работе которого выполняется сканирование последовательности в поиске значения v. Докажите корректность алгоритма с помощью инварианта цикла. Убедитесь, что выбранные инварианты цикла удовлетворяют трем необходимым условиям.

4. Выразите функцию $n^3/1000 - 100n^2 - 100n + 3$ в Θ-обозначениях.

5. Рассмотрим алгоритм линейного поиска (см. задание 3). Для скольких элементов входной последовательности в среднем нужно произвести проверку, если предполагается, что все элементы массива с равной вероятностью могут иметь искомое значение? Что происходит в наихудшем случае? Чему равно время работы алгоритма линейного поиска в среднем и в наихудшем случае (в Θ-обозначениях)? Обоснуйте ваш ответ.

Практические задания

1. Напишите и протестируйте программу, реализующую сортировку вставкой.
2. Напишите и протестируйте программу, реализующую сортировку выбором.

3.* Сортировку вставкой можно представить в виде рекурсивной последовательности следующим образом. Чтобы отсортировать массив $A[1..n]$, сначала нужно выполнить сортировку массива $A[1..n–1]$, после чего в этот отсортированный массив помещается элемент $A[n]$. Напишите и протестируйте программу, реализующую рекурсивный алгоритм сортировки вставками.

Лабораторная работа № 3
Алгоритмы сортировки

Теоретические сведения

Пузырьковая сортировка – сортировка обменами (bubble sort)

Идея

Если при последовательном просмотре пар элементов массива A (a_1, a_2; a_2, a_3 и т.д.) найдется такая пара, что $a_i > a_{i+1}$, то поменять местами a_i и a_{i+1}, после чего продолжить просмотр. Тем самым наибольшее число передвинется на последнее место. Следующие просмотры опять начинать сначала, уменьшая на единицу количество просматриваемых элементов. Массив будет упорядочен после просмотра, в котором участвовали только первый и второй элементы.

Пример

Исходный массив: $A = \langle 5, 2, 4, 6, 1, 3 \rangle$

После первого прохода: $A = \langle 2, 4, 5, 1, 3, \underline{6} \rangle$

После второго прохода: $A = \langle 2, 4, 1, 3, \underline{5, 6} \rangle$

После третьего прохода: $A = \langle 2, 1, 3, \underline{4, 5, 6} \rangle$

После четвертого прохода: $A = \langle 1, 2, \underline{3, 4, 5, 6} \rangle$

После пятого прохода: $A = \langle 1, \underline{2, 3, 4, 5, 6} \rangle$

Алгоритм

BUBBLE_SORT (*A*)

```
 1  for j = A.length downto 2
 2      for i = 1 to j – 1
 3          if A[i] > A[i + 1]
 4              exchange A[i] with A[i + 1]
```

Характеристики

- Время работы алгоритма: $\Theta(n^2)$
- Не использует дополнительную память
- Алгоритм устойчив

Реализация

```
void Bubble_Sort(int A[], int A_length) {
   int i, j;

   for (j=A_length; j >= 2; j--)
      for (i=1; i <= j-1; i++)
        if (A[i] > A[i+1])
           swap(&A[i], &A[i+1]);
}
```

При реализации этого алгоритма сортировки целесообразно написать функцию `swap` для обмена значениями двух переменных (например, элементов массива). При этом надо помнить, что в языке C передача параметров в функцию осуществляется по значению, а не по ссылке, т.е. в функцию необходимо передать не значения переменных, а их адреса.

```
void swap(int *x, int *y) {
   int tmp;

   tmp = *x;
   *x  = *y;
   *y  = tmp;
}
```

Бинарный поиск (binary search)

Идея

Задан упорядоченный массив. Определим местоположение среднего элемента массива и сравним его с ключом поиска.

1. Если они равны, то ключ поиска найден, и функция поиска возвращает индекс этого элемента. В противном случае, задача сокращается на половину элементов массива.
2. Если ключ поиска меньше, чем средний элемент массива, то дальнейший поиск осуществляется в первой половине массива, а если – больше, то – во второй половине.
3. Если ключ поиска не совпадает со средним элементом выбранного подмассива, то алгоритм повторно применяется и сокращает область поиска до четверти исходного массива.

Поиск продолжается до тех пор, пока ключ поиска не станет равным среднему элементу, или пока оставшийся подмассив содержит хотя бы один элемент, не равный ключу поиска.

Пример

Ключ:	10															
Индексы:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Исходный массив:	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
	2	4	6	8	10	12	14									
					10	12	14									
					10											

Ключ:	25														
Индексы:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Исходный массив:	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28
									16	18	20	22	24	26	28
													24	26	28
													24		

Алгоритм

BINARY_SEARCH (*A*, *key*)

```
 1  left = 1
 2  right = A.length
 3  while left ≤ right
 4        middle = (left + right) / 2
 5        if key = A[middle]
 6              return middle
 7        else if key < A[middle]
 8              right = middle – 1
 9        else
10              left = middle + 1
```

Характеристики

- Время работы алгоритма: $\Theta\left(lg\ n\right)$
- Не использует дополнительную память

Реализация

```
void Binary_Search(int A[], int A_length, int key) {
   int left, right, middle;

   left  = 1;
   right = A_length;
   while (left <= right) {
      middle = (low+high) / 2;
      if (key == A[middle])
         return middle;
      else if (key < A[middle])
         right = middle-1;
      else
         left = middle+1;
   }
   return -1;          // key not found
}
```

Теоретические задания

1. Проиллюстрируйте работу алгоритма сортировки методом слияний для массива $A = \langle 3,\ 41,\ 52,\ 26,\ 38,\ 57,\ 9,\ 49 \rangle$.

2. Перепишите процедуру MERGE так, чтобы в ней не использовались сигнальные значения. Сигналом к остановке должен служить тот факт, что все элементы массива L или массива R скопированы обратно в массив A, после чего в этот массив копируются элементы, оставшиеся в непустом массиве.

3. Обратите внимание, что в цикле **while** в строках 5-7 процедуры INSERTION_SORT, используется линейный поиск для просмотра (в обратном порядке) отсортированного подмассива $A[1..j–1]$. Можно ли использовать бинарный поиск вместо линейного, чтобы время работы этого алгоритма в наихудшем случае улучшилось и стало равным в $\Theta\left(n\ lg\ n\right)$?

4. Справедливы ли соотношения $2^{n+1} = O(2^n)$ и $2^{2n} = O(2^n)$?

Практические задания

1. Напишите и протестируйте программу, реализующую пузырьковую сортировку.

2. Напишите и протестируйте программу, реализующую двоичный (би-

нарный) поиск в итерационном виде.

3. Напишите и протестируйте программу, реализующую сортировку слиянием.

4*. Напишите и протестируйте программу, реализующую двоичный (бинарный) поиск в рекурсивном виде. Совет: в качестве параметров функция должна принимать левую и правую границы области поиска.