Для решения поставленных задач была использована реализация языка Prolog –SWI-Prolog.

**Задача №1 (Lab6-1-1.pl):**

Идея метода заключается в сравнение разделенных на группы элементов последовательности, находящихся друг от друга на некотором расстоянии. Количество шагов сортировки (разделение на группы) и шаги (расстояния) в данной программе вычисляются методом, предложенным Дональдом Кнутом. Количество шагов сортировки вычисляется по формуле:

где – количество элементов в списке.

Шаг вычисляется по формуле:

Правила shell\_sort отвечают за сортировку списка: если список пустой, то возвращает пустой список; если список состоит из одного элемента, то возвращается этот элемент, если список состоит из двух элемента, то происходит сравнение этих элементов, и сначала выводится меньший, затем больший; если же список состоит из более чем двух элементов, то запускается сортировка Шелла.

Для вычисления количества шагов сортировки отвечает правило count\_steps. Для вычисления последовательности значений шагов правило list\_steps.

Изначально расстояние равно . На первом шаге каждая группа включает в себя элементы, расположенные друг от друга на расстоянии они сравниваются между собой, и, в случае необходимости, меняются местами. На последующих шагах также происходят проверка и обмен, но расстояние сокращается. Постепенно расстояние между элементами уменьшается, и на проход по массиву происходит в последний раз.

Для запуска программы следует ввести:

shell\_sort([], L).

Тестовые наборы данных и результаты приведены в Таблице 1:

Таблица 1. Тестовые набора данных

|  |  |
| --- | --- |
| Тестовые данные | Результат |
| shell\_sort([], L). | **L** = [] |
| shell\_sort([1], L). | **L** = [1] |
| shell\_sort([1, 2], L). | **L** = [1, 2] |
| shell\_sort([2, 1], L). | **L** = [1, 2] |
| shell\_sort([2, 1, 3], L). | **L** = [1, 2, 3] |
| shell\_sort([1, 2, 6, 23, -1, 1, 3, 1, 3, 4, 5, 6, 8, 9, 12, 3, 5, 6, 89, 1, 2, 4, 5, 7, 2, 3, 0, 1], L). | **L** = [-1, 0, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3,  3, 3, 4, 4, 5, 5, 5, 6, 6, 6, 7, 8, 9, 12, 23, 89] |

**Задача №2 (Lab6-2-1.pl):**

Отличительной особенностью сортировки Хоара является операция разбиения массива на две части относительно опорного элемента. Например, если последовательность требуется упорядочить по возрастанию, то в левую часть будут помещены все элементы, значения которых меньше значения опорного элемента, а в правую элементы, чьи значения больше или равны опорному. Вне зависимости от того, какой элемент выбран в качестве опорного, массив будет отсортирован, но все же наиболее удачным считается ситуация, когда по обеим сторонам от опорного элемента оказывается примерно равное количество элементов. Если длина какой-то из получившихся в результате разбиения частей превышает один элемент, то для нее нужно рекурсивно выполнить упорядочивание, то есть повторно запустить алгоритм на каждом из отрезков.

Правила qsort отвечают за сортировку списка: если список пустой, то возвращает пустой список; если список состоит из одного элемента, то возвращается этот элемент. В противном случае из списка выделяется первый элемент, который берется в качестве опорного, правилом divide список разделяется на две части, поэтому после из рекурсивной сортировки мы можем использовать append для соединения списков, при котором «больший» список дописывается в конец «меньшего».

Для запуска программы следует ввести:

qsort([], L).

Тестовые наборы данных и результаты приведены в Таблице 2:

Таблица 2. Тестовые набора данных

|  |  |
| --- | --- |
| Тестовые данные | Результат |
| qsort([], L). | **L** = [] |
| qsort([1], L). | **L** = [1] |
| qsort([1, 2], L). | **L** = [1, 2] |
| qsort([2, 1], L). | **L** = [1, 2] |
| qsort([2, 1, 3], L). | **L** = [1, 2, 3] |
| qsort([1, 2, 6, 23, -1, 1, 3, 1, 3, 4, 5, 6, 8, 9, 12, 3, 5, 6, 89, 1, 2, 4, 5, 7, 2, 3, 0, 1], L). | **L** = [-1, 0, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3,  3, 3, 4, 4, 5, 5, 5, 6, 6, 6, 7, 8, 9, 12, 23, 89] |

**Задача №3 (Lab6-3-1.pl):**

Идея решения состоит в том, чтобы с помощью правила permutation генерировать списки без повторений и перестановок чисел от 1 до 10, а затем проверять суммы чисел в концах отрезках (соседние), чтобы они не делились ни на 3, ни на 5, и на 7 – правило check\_solve.

Для запуска программы следует ввести:

solve(L).

Полученный результат приведен на рис. 1.

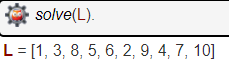


рис. . Решение задания 3.

**Задача №4 (Lab6-4-1.pl):**

За удаление из списка элементов, находящихся на (i \* n) – х местах, где i задается в качестве аргумента, а в n = 1, 2, 3 … отвечают правила delete\_n\_i\_elem: если на вход подается пустой список, то результатом будет пустой список; если на вход подается i = 1, это значит что надо удалить все элементы списка, результатом будет пустой список; если на вход подается такое i, что количество элементов меньше его, то результатом будет исходный список; в остальных случаев с помощью правила get\_number\_delete составляется список индексов, элементы которых следует удалить, после чего с помощью правила delete\_list\_elem они удаляются.

Для запуска программы следует ввести:

delete\_n\_i\_elem([], 1, Result).

Тестовые наборы данных и результаты приведены в Таблице 3:

Таблица 3. Тестовые набора данных

|  |  |
| --- | --- |
| Тестовые данные | Результат |
| delete\_n\_i\_elem([], 1, Result). | **Result** = [] |
| delete\_n\_i\_elem([1, 2, 3, 4, 5, 6, 7, 8, 9], 1, Result). | **Result** = [] |
| delete\_n\_i\_elem([1, 2, 3, 4, 5, 6, 7, 8, 9], 10, Result). | **Result** = [1, 2, 3, 4, 5, 6, 7, 8, 9] |
| delete\_n\_i\_elem([1, 2, 3, 4, 5, 6, 7, 8, 9], 2, Result). | **Result** = [1, 3, 5, 7, 9] |
| delete\_n\_i\_elem([1, 2, 3, 4, 5, 6, 7, 8, 9], 3, Result). | **Result** = [1, 2, 4, 5, 7, 8] |

**Задача №5 (Lab6-2-2.pl):**

В алгоритме сортировки пузырьком содержится два цикла: внутренний перебирает все пары соседних элементов в списке и переставляет их если порядок нарушен; внешний вызывает внутренний цикл до тех пор, пока в списке есть пара элементов, нарушающая порядок.

Если разделить список на сортированную и необработанную части, тогда после обработки внутреннего цикла один элемент (наибольший элемент) гарантированно вытесняется в конец списка (правила move\_max\_to\_end).

Правила bubble\_sort отвечают за сортировку списка:

1. если список пустой, то возвращает пустой список;
2. если список состоит из одного элемента, то возвращается этот элемент.
3. В противном случае из списка запускается сортировка методом пузырька.

Тестовые наборы данных и результаты приведены в Таблице 4:

Таблица 4. Тестовые набора данных

|  |  |
| --- | --- |
| Тестовые данные | Результат |
| bubble\_sort([], Result). | Result = [] |
| bubble\_sort([7], Result). | Result = [7] |
| bubble\_sort([5, 2], Result). | Result = [2, 5] |
| bubble\_sort([2, 5], Result). | Result = [2, 5] |
| bubble\_sort([2, 5, 3, 23, -4], Result). | Result = [-4, 2, 3, 5, 23] |
| bubble\_sort([2, 5, 3, 23, -4, 3, 5, 7, 8, 4, 0, 7, 1, 3, 6, 4, 66, 4, 33, 22, 6, 43, 2, 3, 0], Result). | Result = [-4, 0, 0, 1, 2, 2, 3, 3,  3, 3, 4, 4, 4, 5, 5, 6, 6, 7,  7, 8, 22, 23, 33, 43, 66] |

**Задача №6 (Lab6-4-2.pl):**

За преобразования списка (реверсировать n элементов исходного списка, начиная с i-го) отвечают правила revers:

1. Если на вход подается пустой список, то результатом будет пустой список.
2. Если на вход подается i которая больше, чем длина списка, то возвращается исходный список.
3. Если на вход подается такое N, что количество элементов исходного списка меньше его, то рассчитывается новое N = длина списка – i + 1, то есть изменению подвергнутся все элементы начиная с i-го до конца списка.
4. Если i = 1, то изменения будет происходить с первого элемента списка.
5. В остальных случаях изменения происходят с i-го элемента в исходном списке.

Тестовые наборы данных и результаты приведены в Таблице 5:

Таблица 5. Тестовые набора данных

|  |  |
| --- | --- |
| Тестовые данные | Результат |
| revers([], 1, 2, Result). | Result = [] |
| revers([1, 2, 3, 4, 5], 6, 2, Result). | Result = [1, 2, 3, 4, 5] |
| revers([1, 2, 3, 4, 5], 4, 20, Result). | Result = [1, 2, 3, 5, 4] |
| revers([1, 2, 3, 4, 5], 1, 5, Result). | Result = [5, 4, 3, 2, 1] |
| revers([1, 2, 3, 4, 5], 2, 3, Result). | Result = [1, 4, 3, 2, 5] |

**Задача №7 (Lab6-2-3.pl):**

Правила selection\_sort отвечают за сортировку списка:

1. если список пустой, то возвращает пустой список;
2. если список состоит из одного элемента, то возвращается этот элемент,
3. в ином случае запускается сортировка методом прямого выбора.

Алгоритм сортировки выбором постепенно выбирает из неупорядоченной части списка минимальные элементы и добавляет их в отсортированную часть. Для поиска минимального элемента используется правило min\_list, для удаления элемента с заданным значением delete\_single\_element.

Список разделяется на упорядоченную и необработанную части. Из неупорядоченной части выбирается минимальный элемент, его значение удаляется из списка. В результате рекурсивной обработки остальных элементов формируется новый список, в начало которого необходимо добавить выделенный ранее элемент. Если исходный список пуст, значит все элементы были отсортированы – необходимо вернуть их в качестве результата.

Тестовые наборы данных и результаты приведены в Таблице 6:

Таблица 6. Тестовые набора данных

|  |  |
| --- | --- |
| Тестовые данные | Результат |
| selection\_sort([], Result). | Result = [] |
| selection\_sort([5], Result). | Result = [5] |
| selection\_sort([1, -2, 0, 1], Result). | Result = [-2, 0, 1, 1] |
| selection\_sort([1, -2, 0, 1, 6, 3, 9, 6, 1, 7, 3, 5, 77, 4, 3, 2, 0, 8, 5, -3, -2, 12, 32, 76, 4], Result). | Result = [-3, -2, -2, 0, 0, 1, 1,  1, 2, 3, 3, 3, 4, 4, 5, 5, 6, 6,  7, 8, 9, 12, 32, 76, 77] |

**Задача №8 (Lab6-4-3.pl):**

За вставку подсписка в список, начиная с i-го элемента, отвечает правило insert\_sublist:

1. Если подсписок пустой, то результатом будет исходный список.
2. Если исходный список пустой, то результатом будет подсписок.
3. Если элемент, с которого следует начинать вставлять подсписок больше, чем длина исходного списка, то подсписок вставляется в конец исходного подсписка.
4. Если i = 1, то подсписок вставляется в начало исходного списка.
5. Иначе происходит вставка подсписка в список, начиная с i-го элемента.

Тестовые наборы данных и результаты приведены в Таблице 7:

Таблица 7. Тестовые набора данных

|  |  |
| --- | --- |
| Тестовые данные | Результат |
| insert\_sublist([1, 2, 3], [], 0, Result). | Result = [1, 2, 3] |
| insert\_sublist([], [1, 2], 0, Result). | Result = [1, 2] |
| insert\_sublist([1, 2, 3, 4, 5], [0, 0, 0], 10, Result). | Result = [1, 2, 3, 4, 5, 0, 0, 0] |
| insert\_sublist([1, 2, 3, 4, 5], [0, 0, 0], 1, Result). | Result = [0, 0, 0, 1, 2, 3, 4, 5] |
| insert\_sublist([1, 2, 3, 4, 5], [0, 0, 0], 2, Result). | Result = [1, 0, 0, 0, 2, 3, 4, 5] |
| insert\_sublist([1, 2, 3, 4, 5], [0, 0, 0], 4, Result). | Result = [1, 2, 3, 0, 0, 0, 4, 5] |