**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Лабораторная работа №4

по курсу

«Логическое и функциональное программирование»

Выполнил:

студент группы ИКПИ-14

Хохлов Т.В.

Принял:

доцент кафедры ПИиВТ

Ерофеев С.А.

Санкт-Петербург

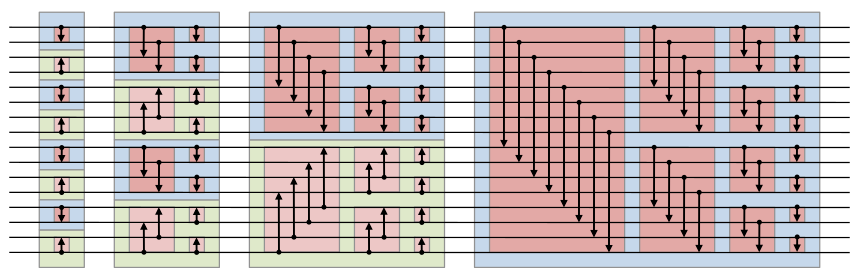
2023 г.

**Цель работы**

Разработать программу на языке Turbo Prolog для сдедующей задачи:

Дана битоническая последовательность - конечный упорядоченный набор из чисел, в котором они сначала монотонно возрастают, а затем монотонно убывают. Необходимо упорядочить данный набор чисел с помощью битонной сортировки.

**Описание алгоритма**

****

На рисунке изображена битонная сортировочная сеть для 16 элементов, которая сортирует множество по возрастанию. Стрелки изображают [компараторы](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%BE%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D0%B0%D1%80%D0%B0%D1%82%D0%BE%D1%80), которые сравнивают два числа и при необходимости меняют их местами таким образом, чтобы направление стрелки указывало на большее число.

Красные прямоугольники скомбинированы в зеленые и голубые прямоугольники. В синих прямоугольниках стрелки компараторов направлены вниз (создают возрастающие последовательности), в зеленых — вверх (создают убывающие последовательности). Каждый из таких прямоугольников имеет одинаковую структуру: красный прямоугольник применяется ко всей последовательности, затем к каждой половине полученных результатов и так далее. Если на входы такого прямоугольника подается битонная последовательность, то на выходе она преобразуется в полностью отсортированную. Объединенные результаты синего и зеленого прямоугольника является битонной последовательностью.

Каждый столбец синих и зеленых прямоугольников принимает N отсортированных последовательностей (на самом первом шаге 16 отсортированных последовательностей, состоящих из 1 элемента) и преобразует их в N/2 отсортированных последовательностей.

**Реализация**

Введённый массив чисел будет хранится в виде целочисленного списка.

Для начала ввода целочисленного массива, необходимо указать его размерность. Обязательное требование для возможности реализации битонной сортировки – размерность массива 2n. Именно поэтому данная программа запрашивает размерность массива в виде степени двойки.

Предикат checksize отвечает за проверку правильности ввода (размерность массива не может быть отрицательной).

Предикат powOfTwo получает конкретное количество элементов в массиве. Проще говоря данный предикат возводит двойку в степень N введённую пользователем.

Предикат inputList отвечает за непосредственный ввод целочисленного списка.

Предикаты replace\_at\_index и get\_element – вспомогательные предикаты для замены элемента списка по индексу и получения значения по индексу. Данные предикаты являются частью предиката swap для смены позиций двух элементов в списке соответственно.

Предикат runway – главные предикат данной программы. В нём реализована логика одного «пробега» по списку с определенным отступом. В то время как в предикате loop реализован полный проход по всему списку со всеми возможными отступами.

**Используемые предикаты**

readSize(integer) – считывание размерности списка (2N)

checkSize(integer) – проверка размерности списка

powOfTwo(integer, integer) – вычисление количества элементов в списке

inputList(integer, list) – пользовательский ввод списка

printList(list) – вывод списка на экран

replace\_at\_index(integer,integer,list,list) – замена элемента списка по индексу

get\_element(integer,list,integer) – получение значения элемента по индексу

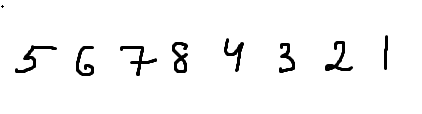
swap(integer, integer, list, list) – смена значений списка по индексам

runway(integer,integer, list,list) – «пробег» по списку в рамках выбранного отступа

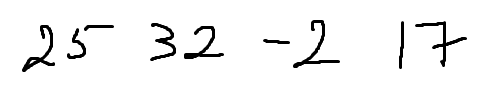
loop(integer,integer,list,list) – цикл изменения отступов для «пробега»

**Тестирование**

Возможный список целочисленных чисел Результат:



Возможный список целочисленных чисел Результат:

**Выводы**

В ходе проведенной лабораторной работы я разработал программу на языке Turbo Prolog для сортировки битонных последовательностей. Было проведено тестирование отдельных предикатов и всей программы в целом. Программа полностью удовлетворяет заданным требованиям.

# Исходный код программы

**%trace**

**domains**

**list = integer\***

**predicates**

**readSize(integer)**

**checkSize(integer)**

**powOfTwo(integer, integer)**

**inputList(integer, list)**

**printList(list)**

**split(integer, list, list, list)**

**listFirst(integer, list, list, list)**

**replace\_at\_index(integer,integer,list,list)**

**get\_element(integer,list,integer)**

**swap(integer, integer, list, list)**

**runway(integer,integer, list,list)**

**loop(integer,integer,list,list)**

**clauses**

**readSize(X) :-**

**clearwindow,**

**write("Enter value "),nl,**

**readint(X),!;**

**readSize(X).**

**checkSize(X) :-**

**X > 0;**

**write("Negative power"),!.**

**powOfTwo(0, 1).**

**powOfTwo(X, RES) :-**

**X > 0,**

**X1 = X - 1,**

**powOfTwo(X1, TempRES),**

**RES = TempRES \* 2.**

**inputList(0, []).**

**inputList(Size, [H|T]) :-**

**clearwindow,**

**write(Size), write(" values left"),nl,**

**readint(H),**

**Size1=Size-1,**

**inputList(Size1, T).**

**split(Size,[H|T],Left,Right) :-**

**Half = Size / 2,**

**listFirst(Half, [H|T], Left, Right).**

**listFirst(0, Tail, [], Tail).**

**listFirst(Half, [H|T], [H|Tl], Tail) :-**

**Half1 = Half - 1,**

**listFirst(Half1, T, Tl, Tail).**

**swap(Index1,Index2,List,NewList) :-**

**get\_element(Index1, List, Value1),**

**get\_element(Index2, List, Value2),**

**replace\_at\_index(Index1, Value2, List, TempList),**

**replace\_at\_index(Index2, Value1, TempList, NewList).**

**replace\_at\_index(0, NewValue, [\_|Tail], [NewValue|Tail]).**

**replace\_at\_index(Index, NewValue, [Head|Tail], [Head|NewTail]) :-**

**Index > 0,**

**NewIndex = Index - 1,**

**replace\_at\_index(NewIndex, NewValue, Tail, NewTail).**

**get\_element(0, [Head|\_], Head).**

**get\_element(Index, [\_|Tail], Element) :-**

**Index > 0,**

**NewIndex = Index - 1,**

**get\_element(NewIndex, Tail, Element).**

**runway(-1,\_,List,List).**

**runway(Iters, Offset, List, Out) :-**

**NewIters = Iters - 1,**

**Index1 = Iters,**

**Index2 = Index1 + Offset,**

**% write(Index1),**

**% write(Index2),nl,**

**get\_element(Index1,List,Val1),**

**get\_element(Index2,List,Val2),**

**% write(Val1),write(" "),**

**% write(Val2),write(" "),nl,**

**Val1 <= Val2,**

**% write(Offset), write(" "), write(Iters),nl,**

**% readchar(\_),**

**runway(NewIters, Offset, List, Out);**

**NewIters = Iters - 1,**

**Index1 = Iters,**

**Index2 = Index1 + Offset,**

**% write(Index1),nl,**

**% write(Index2),nl,**

**get\_element(Index1,List,Val1),**

**get\_element(Index2,List,Val2),**

**% write(Val1),write(" "),**

**% write(Val2),write(" "),nl,**

**Val1 > Val2,**

**% write(Offset), write(" "), write(Iters),nl,**

**% readchar(\_),**

**swap(Index1,Index2,List,NewList),**

**runway(NewIters, Offset, NewList, Out).**

**loop(Size, 0.5, List, Out) :-**

**Iters = Size - 2,**

**runway(Iters, 1, List, Out).**

**loop(Size, Offset, List, Out) :-**

**Iters = Size - Offset - 1,**

**runway(Iters, Offset, List, TempList),**

**NewOffset = Offset/2,**

**loop(Size,NewOffset,TempList,Out).**

**printList([]).**

**printList([H|T]) :-**

**write(H), write(" "),**

**printList(T).**

**goal**

**clearwindow,**

**%runway(6, 1, [1,2,3,4,8,7,6,5], Out),**

**%loop(8, 4, [8,7,6,5,1,2,3,4], Out),**

**%printList(Out).**

**readSize(Size),**

**checkSize(Size),**

**powOfTwo(Size, RES),**

**%write(RES).**

**inputList(RES, List),**

**clearwindow,**

**Offset = RES / 2,**

**loop(RES,Offset,List,NewList),**

**printList(NewList).**