Міністерство освіти та науки України

Національний технічний університет України “КПІ ім. Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

**ЗВІТ**

про виконання

Лабораторної роботи №2

з дисципліни:

«**Декларативне програмування**»

**Тема: «Опис простих рекурсивних функцій в мові Lisp»**

**Варіант 6**

**Виконав:** студент групи ІП-51

Зарічковий Олександр Анатолійович

**Перевірив**: доц. Баклан І. В.

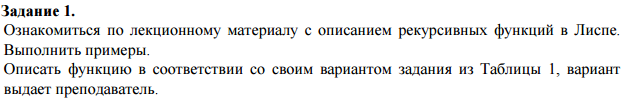
Київ 2017

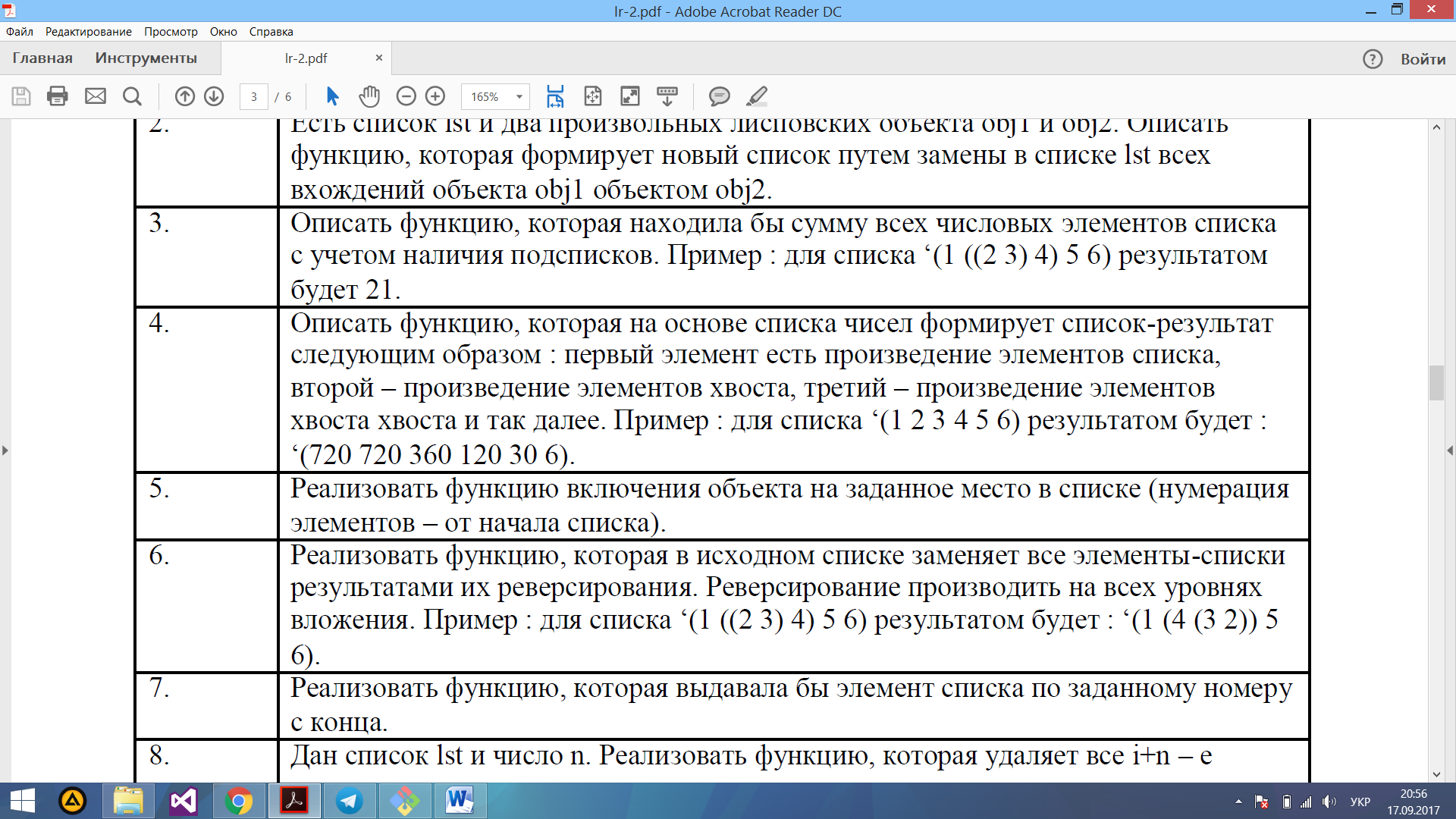
**Цель работы:**

Целью работы является изучение основных правил написания рекурсивных функций в функциональном языке и изучение основных методов разработки функциональных программ с позиций Строго Функционального Языка.

**Основние задачи:**

* На примере GNU Common Lisp‟а (GCLisp'а) научиться формулировать условие завершения рекурсии, описывать формирование результата функции и новых значений аргументов для рекурсивного вызова
* Получить практические навыки работы со списочными структурами в выбранной реализации языка Лисп
* Освоить приемы нисходящего и восходящего проектирования функциональных программ
* Научиться выделять основные и вспомогательные функции с учетом разбиения задачи на подзадачи
* Овладеть приемами использования накапливающих параметров во вспомогательных функциях
* Ознакомиться с упреждающим использованием результата вызова функции.





**Код програми:**

;TASK #1

(DEFUN RECURSIVE\_REVERSE

(LIST\_TO\_REVERSE)

(COND ((NULL LIST\_TO\_REVERSE) '())

((ATOM LIST\_TO\_REVERSE) (LIST LIST\_TO\_REVERSE))

((ATOM (CAR LIST\_TO\_REVERSE)) (APPEND (RECURSIVE\_REVERSE (CDR LIST\_TO\_REVERSE)) (RECURSIVE\_REVERSE (CAR LIST\_TO\_REVERSE))))

(T (APPEND (RECURSIVE\_REVERSE (CDR LIST\_TO\_REVERSE)) (LIST (RECURSIVE\_REVERSE (CAR LIST\_TO\_REVERSE)))))

)

)

(DEFUN REVERSE\_ONLY\_LISTS

(LST)

(COND ((NULL LST) '())

((ATOM LST) (LIST LST))

((ATOM (CAR LST)) (APPEND (REVERSE\_ONLY\_LISTS (CAR LST)) (REVERSE\_ONLY\_LISTS (CDR LST))))

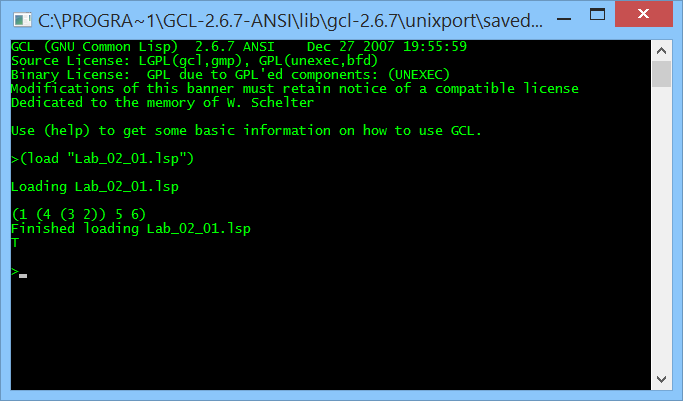
(T (APPEND (LIST (RECURSIVE\_REVERSE (CAR LST))) (REVERSE\_ONLY\_LISTS (CDR LST))))

)

)

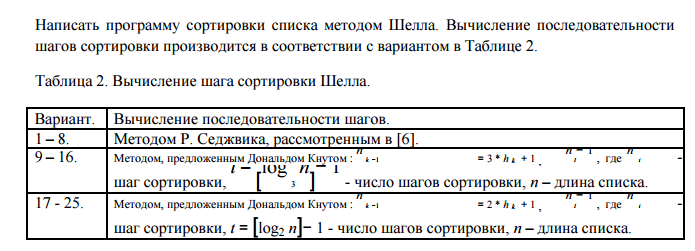
(print (REVERSE\_ONLY\_LISTS '(1 ((2 3) 4) 5 6)))

**Скріншот програми:**



В цій задачі я створив функцію яка реверсує рекурсивно списки та функцію яка проходить по 1 рівні заданого списку та виликає попердню функцію.

**Задание 2**



**Код програми:**

;TASK #2

(DEFUN insertion (lst x)

(COND ((NULL lst) (LIST x))

((> (CAR lst) x) (CONS x lst))

(t (CONS (CAR lst) (insertion (CDR lst) x)))))

(DEFUN isort (x &optional (s nil))

(COND ((NULL x) s)

(t (isort (CDR x) (insertion s (CAR x))))))

(DEFUN shell (lst gap finalLIST)

(COND ((NULL lst) finalLIST)

(T (COND ((> gap (LENGTH lst)) (APPEND finalLIST (isort lst)))

(T (APPEND finalLIST (isort (subseq lst 0 gap)) (shell (subseq lst gap (LENGTH lst)) gap finalLIST)))))

))

(DEFUN shellSort (lst gaps)

(COND ((NULL (CDR gaps)) (shell lst (CAR gaps) '()))

(T (shellSort (shell lst (CAR gaps) '()) (CDR gaps)))))

(DEFUN SedgewickGaps (len finalLIST)

'(1 8 23 77 281 1073 4193 16577 65921 262913 1050113 4197377 16783361

67121153 268460033 1073790977 4295065601 17180065793 68719869953 274878693377

1099513200641 4398049656833 17592192335873 70368756760577)

)

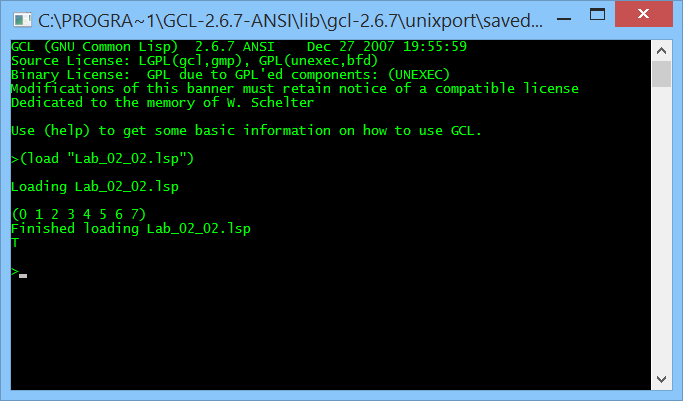
(DEFUN sortWithShellAndSedgewick (lst)

(shellSort lst (SedgewickGaps (LENGTH lst) '(1)))

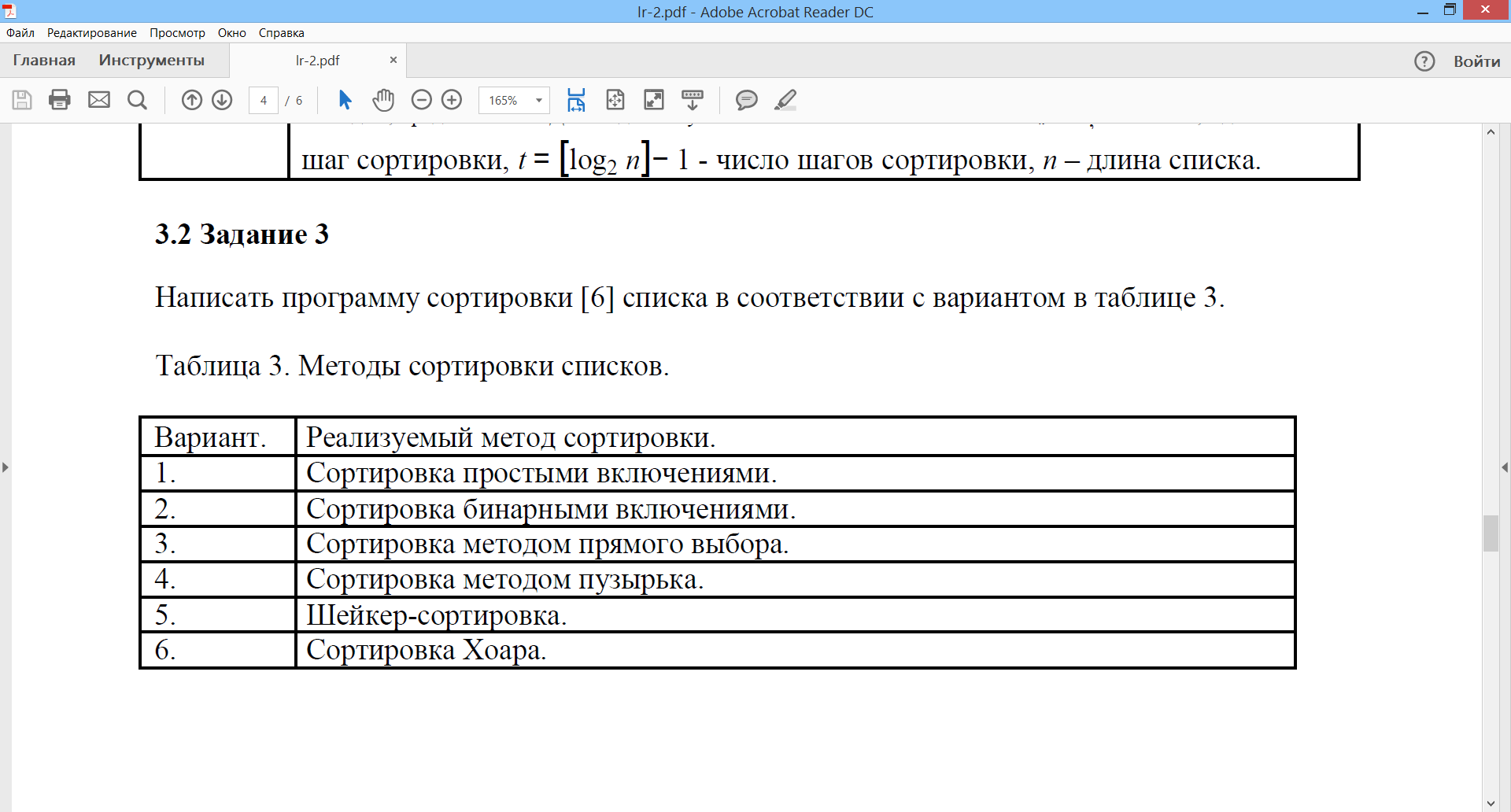
)

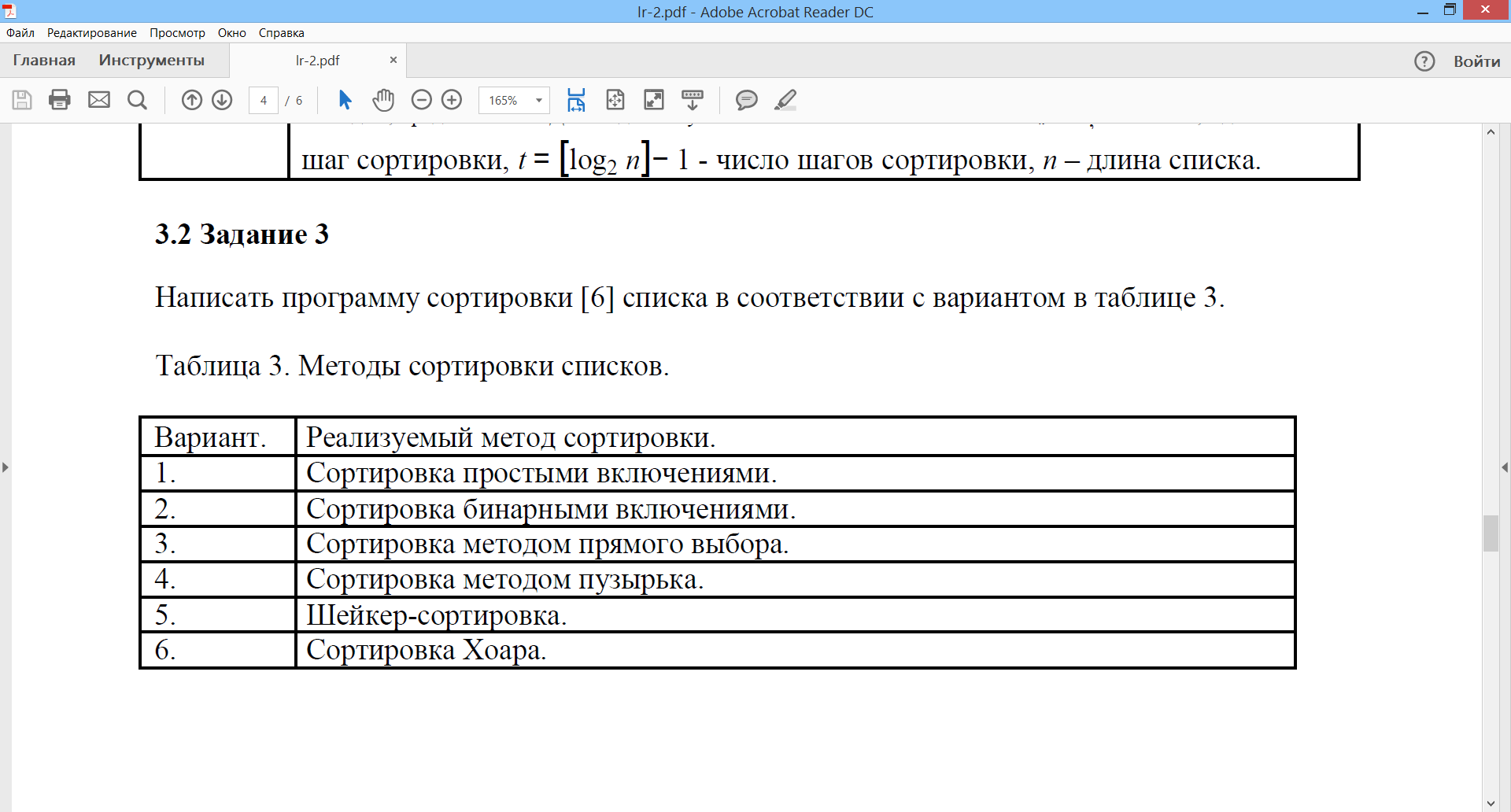
(print (sortWithShellAndSedgewick '(7 6 5 4 3 1 2 0)))

**Скріншот програми:**



В цьому завданні я реалізував сортування Шелла за допомогою рекурсивних викликів ряду функцій. Також я реалізував функцію я повертає ряд Седжвіка, яка використовується в сортуванні Шелла.





**Код програми:**

;TASK #3

(DEFUN LIST< (a b)

(COND

((or (NULL a)(NULL b)) NIL)

(( < a (CAR b)) (LIST< a (CDR b)))

(t(CONS (CAR b) (LIST< a (CDR b))))))

(DEFUN LIST>= (a b)

(COND

((or (NULL a)(NULL b)) NIL)

(( >= a (CAR b)) (LIST>= a (CDR b)))

(T (CONS (CAR b) (LIST>= a (CDR b))))))

(DEFUN qsort (L)

(COND

((NULL L) nil)

(T (APPEND

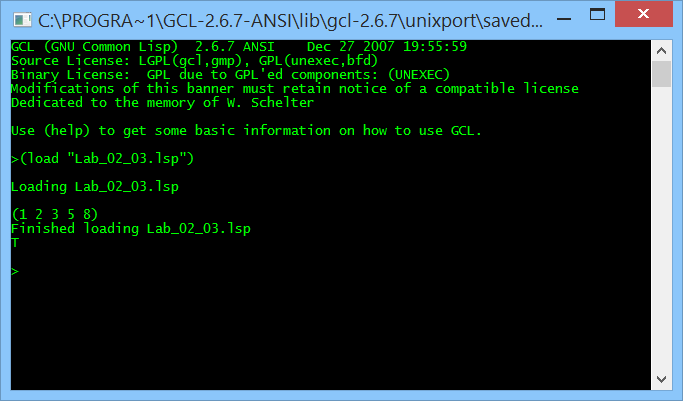
(qsort (LIST< (CAR L) (CDR L)))

(CONS (CAR L) nil)

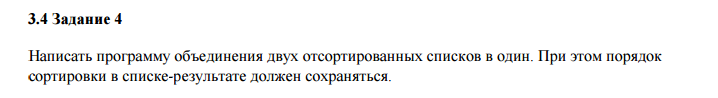
(qsort (LIST>= (CAR L) (CDR L)))))))

(print (qsort '(1 5 3 8 2)))

**Скріншот програми:**



В цьому завдані я дві функції, які розбивають список відносно опорного елемнту і зарахунок них реалізує сортування Хоара, яка писується як рекусивний вилик сортування Хоара для списків зліва та справа опорого елемента.



**Код програми:**

;TASK #4

(DEFUN merge\_LISTs

(LIST1 LIST2)

(COND ((NULL LIST1) LIST2)

((NULL LIST2) LIST1)

((> (CAR LIST1) (CAR LIST2)) (CONS (CAR LIST2) (merge\_LISTs LIST1 (CDR LIST2))))

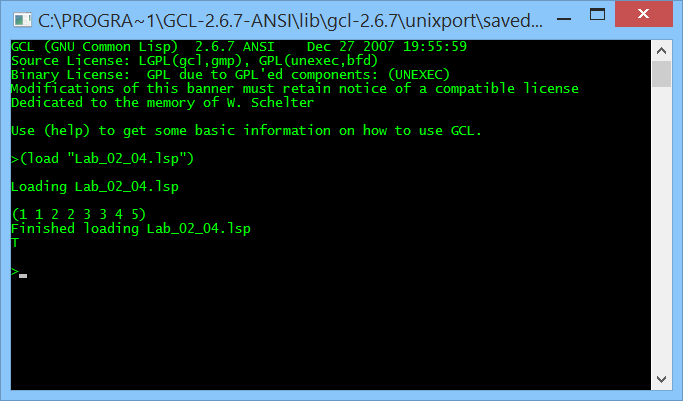
(T (CONS (CAR LIST1) (merge\_LISTs (CDR LIST1) LIST2)))

)

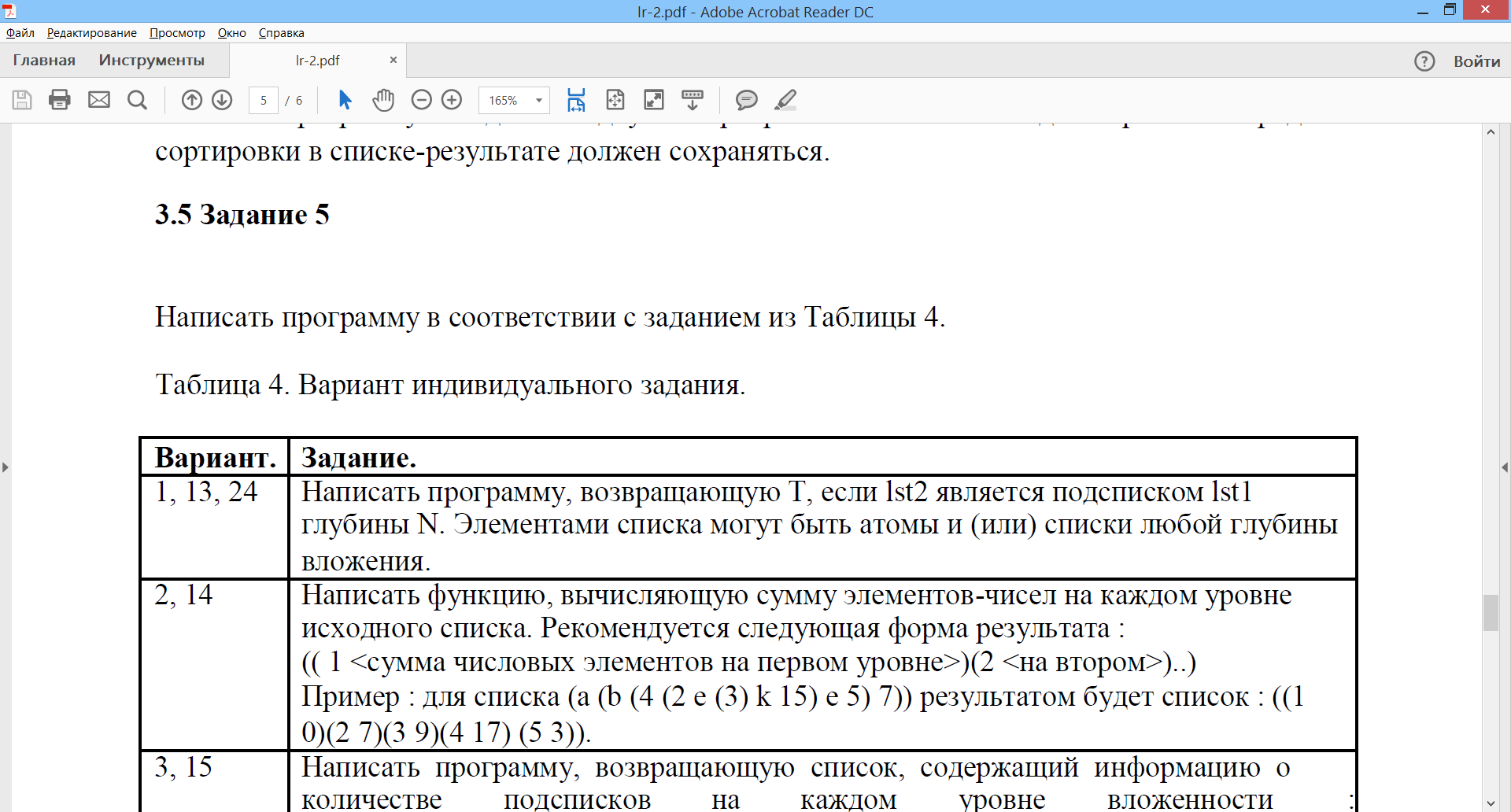
)

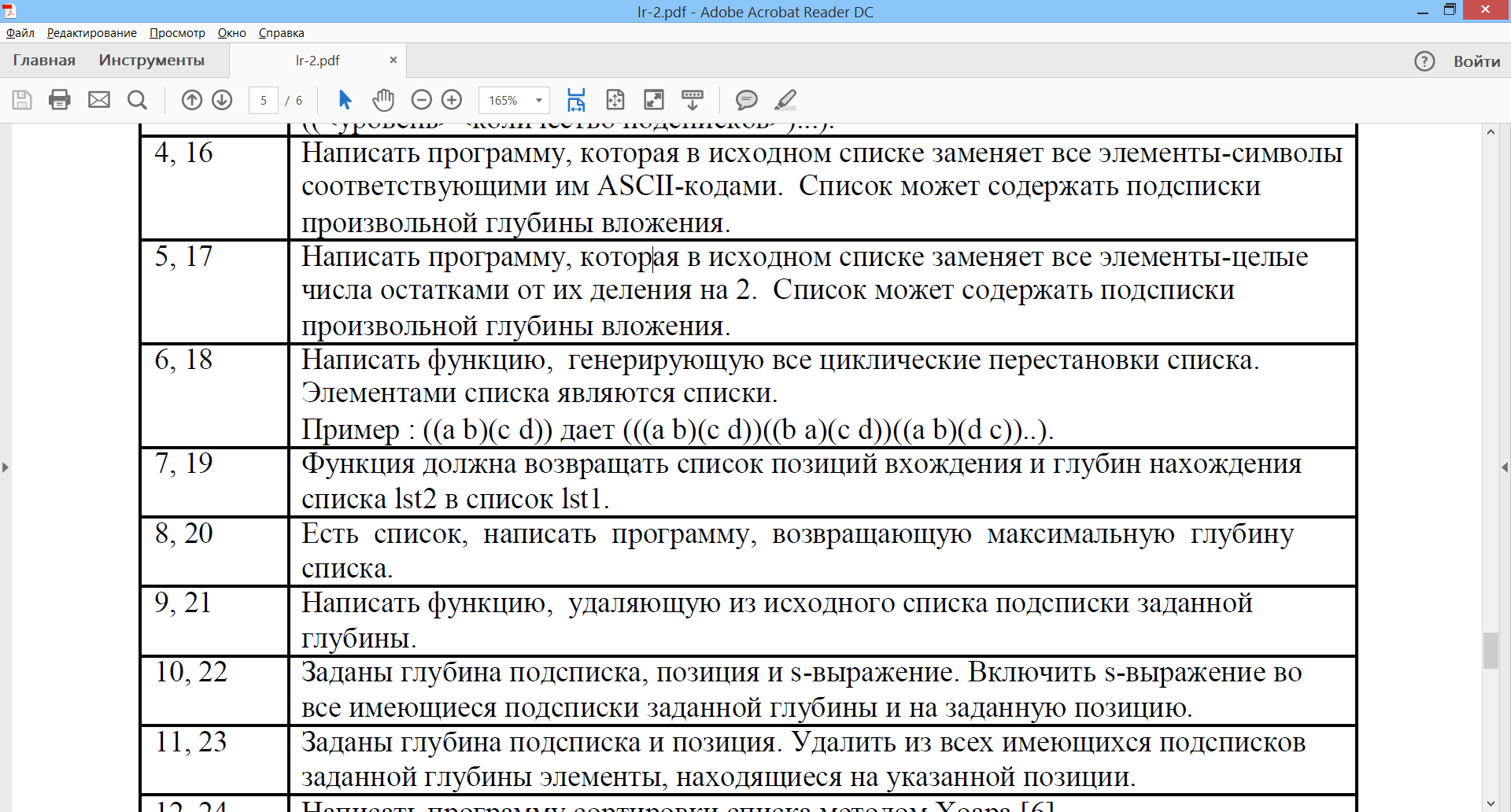
(print (merge\_LISTs '(1 2 3) '(1 2 3 4 5)))

**Скріншот програми:**



В цьому завдані реалізува функію злиття двох відсортованих списків за допомогою рекурсивного виклику злиття хвостів двох списків.





**Код програми:**

;TASK #5

(DEFUN rotate(l n)

(APPEND (NTHCDR n l) (butlast l (- (LENGTH l) n))))

(DEFUN rotate\_all

(lst n cum\_sum)

(COND ((NULL lst) (LIST cum\_sum))

((NULL (NTH n (CAR lst))) nil)

(T (APPEND (rotate\_all (CDR lst) 0 (APPEND cum\_sum (LIST (rotate (CAR lst) n))))

(rotate\_all lst (+ n 1) cum\_sum)))

)

)

(DEFUN rotate\_all\_wraper

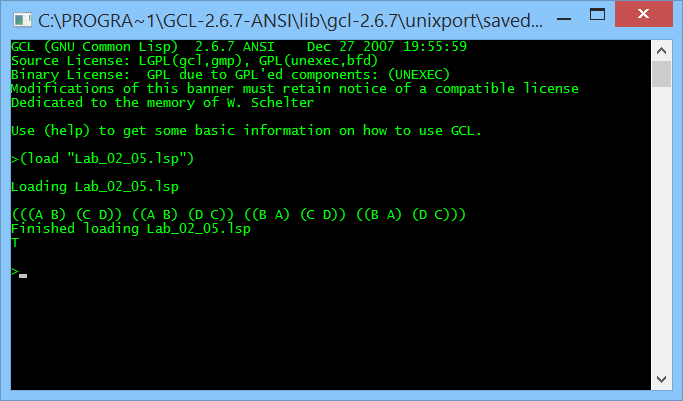
(lst)

(rotate\_all lst 0 nil))

)

(print (rotate\_all\_wrapper '((a b) (c d))))

**Скріншот програми:**



В цьому завдані реалізува яка виконую циклічну перестановку списку. На основі даної функції я реалізував рекурсивну функцію, яка створює список всіх можливих циклічних перестановок даного списку списків.

**Листинг всієї програми:**

;gnu clisp 2.49

;TASK #1

(DEFUN RECURSIVE\_REVERSE

(LIST\_TO\_REVERSE)

(COND ((NULL LIST\_TO\_REVERSE) '())

((ATOM LIST\_TO\_REVERSE) (LIST LIST\_TO\_REVERSE))

((ATOM (CAR LIST\_TO\_REVERSE)) (APPEND (RECURSIVE\_REVERSE (CDR LIST\_TO\_REVERSE)) (RECURSIVE\_REVERSE (CAR LIST\_TO\_REVERSE))))

(T (APPEND (RECURSIVE\_REVERSE (CDR LIST\_TO\_REVERSE)) (LIST (RECURSIVE\_REVERSE (CAR LIST\_TO\_REVERSE)))))

)

)

(DEFUN REVERSE\_ONLY\_LISTS

(LST)

(COND ((NULL LST) '())

((ATOM LST) (LIST LST))

((ATOM (CAR LST)) (APPEND (REVERSE\_ONLY\_LISTS (CAR LST)) (REVERSE\_ONLY\_LISTS (CDR LST))))

(T (APPEND (LIST (RECURSIVE\_REVERSE (CAR LST))) (REVERSE\_ONLY\_LISTS (CDR LST))))

)

)

(print (REVERSE\_ONLY\_LISTS '(1 ((2 3) 4) 5 6)))

;TASK #2

(DEFUN insertion (lst x)

(COND ((NULL lst) (LIST x))

((> (CAR lst) x) (CONS x lst))

(t (CONS (CAR lst) (insertion (CDR lst) x)))))

(DEFUN isort (x &optional (s nil))

(COND ((NULL x) s)

(t (isort (CDR x) (insertion s (CAR x))))))

(DEFUN shell (lst gap finalLIST)

(COND ((NULL lst) finalLIST)

(T (COND ((> gap (LENGTH lst)) (APPEND finalLIST (isort lst)))

(T (APPEND finalLIST (isort (subseq lst 0 gap)) (shell (subseq lst gap (LENGTH lst)) gap finalLIST)))))

))

(DEFUN shellSort (lst gaps)

(COND ((NULL (CDR gaps)) (shell lst (CAR gaps) '()))

(T (shellSort (shell lst (CAR gaps) '()) (CDR gaps)))))

(DEFUN SedgewickGaps (len finalLIST)

'(1 8 23 77 281 1073 4193 16577 65921 262913 1050113 4197377 16783361

67121153 268460033 1073790977 4295065601 17180065793 68719869953 274878693377

1099513200641 4398049656833 17592192335873 70368756760577)

)

(DEFUN sortWithShellAndSedgewick (lst)

(shellSort lst (SedgewickGaps (LENGTH lst) '(1)))

)

(print (sortWithShellAndSedgewick '(7 6 5 4 3 1 2 0)))

;TASK #3

(DEFUN LIST< (a b)

(COND

((or (NULL a)(NULL b)) NIL)

(( < a (CAR b)) (LIST< a (CDR b)))

(t(CONS (CAR b) (LIST< a (CDR b))))))

(DEFUN LIST>= (a b)

(COND

((or (NULL a)(NULL b)) NIL)

(( >= a (CAR b)) (LIST>= a (CDR b)))

(T (CONS (CAR b) (LIST>= a (CDR b))))))

(DEFUN qsort (L)

(COND

((NULL L) nil)

(T (APPEND

(qsort (LIST< (CAR L) (CDR L)))

(CONS (CAR L) nil)

(qsort (LIST>= (CAR L) (CDR L)))))))

(print (qsort '(1 5 3 8 2)))

;TASK #4

(DEFUN merge\_LISTs

(LIST1 LIST2)

(COND ((NULL LIST1) LIST2)

((NULL LIST2) LIST1)

((> (CAR LIST1) (CAR LIST2)) (CONS (CAR LIST2) (merge\_LISTs LIST1 (CDR LIST2))))

(T (CONS (CAR LIST1) (merge\_LISTs (CDR LIST1) LIST2)))

)

)

(print (merge\_LISTs '(1 2 3) '(1 2 3 4 5)))

;TASK #5

(DEFUN rotate

(l n)

(APPEND (NTHCDR n l) (butlast l (- (LENGTH l) n)))

)

(DEFUN rotate\_all

(lst n cum\_sum)

(COND ((NULL lst) (LIST cum\_sum))

((NULL (NTH n (CAR lst))) nil)

(T (APPEND (rotate\_all (CDR lst) 0 (APPEND cum\_sum (LIST (rotate (CAR lst) n))))

(rotate\_all lst (+ n 1) cum\_sum)))

)

)

(DEFUN rotate\_all\_wrapper

(lst)

(rotate\_all lst 0 nil)

)

(print (rotate\_all\_wrapper '((a b) (c d))))

**Висновок:**

В цій лабораторній роботі я опанував базовий механізм виклику та опису рекурсивних функцій у інтерпретованій мові Lisp, навчилався користуватися інтерпретатором GCL та завантажувати з нього файли з розширенням \*.lsp, також опанував деякі продвинуті функції роботи зі списками.