GENERAL CONFEDERATION OF LABOR OF VIETNAM **TON DUC THANG UNIVERSITY FACULTY OF INFORMATION TECHNOLOGY**



**SOFTWARE ENGINEERING**

**PHÂN TÍCH THIẾT KẾ**

**VÀ GIẢI THUẬT**

*Instructing Lecturer*:**TS Nguyễn Chí Thiện**

*Student’s name*: **THI NGỌC PHÚ– 518H0044**

**NGUYỄN KHÁNH VINH-518H0076**

Class**:** **18H50201**

Course**:** **22**

**HO CHI MINH CITY, May 2020**

i

# **ACKNOWLEDGEMENT**

In order to make this report complete and achieve good results, we have received the support and assistance of many teachers and classmates.

With deep affection, sincerity, we express deep gratitude to all individuals and agencies who have helped us in our study and research.

First of all, I would like to express a special appreciation to Ton Duc Thang.

University’s teachers for their conscientious guidance and advices throughout the last semester by gave me their modern outlook and meticulous supervision to carry out the job perfectly.

Especially we would like to send our sincere thanks to Mr. Dang Minh Thang has paid attention, help, guide us to complete the report well over the past time.

We would like to express our sincere gratitude to the leadership of Ton Duc Thang University for supporting, helping and facilitating us to complete the report well during the study period.

With limited time and experience, this report cannot avoid mistakes. We are looking forward to receiving advice and comments from teachers so that we can improve our awareness, better serve the practical work later.

We sincerely thank you!

ii

# **THE PROJECT WAS COMPLETED AT TON DUC THANG UNVERSITY**

I pledge that this is a product of our own project and is under the guidance of Mr. Nguyen Khanh Tung. The content of research, results in this subject is honest and not published in any form before. The data in the tables used for the analysis, comment, and evaluation were collected by the authors themselves from various sources indicated in the reference section. In addition, many comments and assessments as well as data from other authors and organizations have been used in the project, with references and annotations.

**If any fraud is found, I am fully responsible for the content of my project.**

Ton Duc Thang University is not involved in any copyright infringement or copyright infringement in the course of implementation (if any).

*Ho Chi Minh City, May 24th 2020*

*Author*

*(Sign full name)*

*Signed*

*Thi Ngọc Phú*

iii

# **EVALUATION OF INSTRUCTING LECTURER**

**Confirmation of the instructor**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Ho Chi Minh City, May 24th 2020*

(sign full name)

**The assessment of the teacher marked**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Ho Chi Minh City, May 24th 2020*

(sign full name)

**TABLE OF CONTENTS**

[**ACKNOWLEDGEMENT** 2](#_Toc56692995)

[**THE PROJECT WAS COMPLETED AT TON DUC THANG UNVERSITY** 3](#_Toc56692996)

[**EVALUATION OF INSTRUCTING LECTURER** 4](#_Toc56692997)

[I. Giới thiệu 3](#_Toc56692998)

[II. Các thuật toán đã học 3](#_Toc56692999)

[2.1 Brute force 3](#_Toc56693000)

[A. Ý tưởng cơ bản : 3](#_Toc56693001)

[2.1.1 Selection sort 3](#_Toc56693002)

[a. Ý tưởng 3](#_Toc56693003)

[b. Khai triển 3](#_Toc56693004)

[c. Demo 4](#_Toc56693005)

[2.1.2 Buble sort 5](#_Toc56693006)

[a. Ý tưởng 5](#_Toc56693007)

[b. Khai triển 5](#_Toc56693008)

[c. Demo 6](#_Toc56693009)

[2.1.3 Sequential search 7](#_Toc56693010)

[b. Khai triển 8](#_Toc56693011)

[c. Demo 8](#_Toc56693012)

[2.2 Divide-and-conquer 9](#_Toc56693013)

[2.2.1 Mergesort 10](#_Toc56693014)

[a. Ý tưởng 10](#_Toc56693015)

[b. Khai triển 10](#_Toc56693016)

[c. Demo 11](#_Toc56693017)

[2.2.2 Quicksort 13](#_Toc56693018)

[a. Ý tưởng 13](#_Toc56693019)

[b. Khai triển 13](#_Toc56693020)

[c. Demo 14](#_Toc56693021)

[2.4 Dynamic Programming 16](#_Toc56693022)

[2.4.1 Coin-row problem 16](#_Toc56693023)

[a. Ý tưởng 16](#_Toc56693024)

[b. Khai triển 16](#_Toc56693025)

[c. Demo 17](#_Toc56693026)

[2.4.2 Change-making problem 19](#_Toc56693027)

[a. Ý tưởng 19](#_Toc56693028)

[b. Khai triển 19](#_Toc56693029)

[c. Demo 19](#_Toc56693030)

**TABLE OF FIRGURES**

**FIRGURE**

1. Giới thiệu
2. Các thuật toán đã học
   1. Brute force
3. Ý tưởng cơ bản :

* Giải quyết vấn đề trực tiếp
  + 1. Selection sort

1. Ý tưởng

* Bắt đầu bằng cách quét toàn bộ danh sách các phần tử để tìm phần tử nhỏ nhất và tráo chúng với phần tử đầu tiên, đặt phần tử nhỏ nhất ở vị trí cuối cùng trong danh sách đã được sếp. Sau đó quét tiếp danh sách tìm phần tử thứ 2 để tìm phần tử nhỏ nhất trong số n – 1 phần tử cuối cùng và tráo chúng với phần tử thứ 2, đặt tiếp phần tử thứ 2 nhỏ nhất vào cuối cùng.

1. Khai triển

# Sorts a given array by selection sort

# Input: An array A[0..n − 1] of orderable elements

# Output: Array A[0..n − 1] sorted in nondecreasing order

def SelectionSort(A):

    n = len(A)

    for i in range(0, n - 1):

        min = i

        for j in range(i + 1, n):

            if A[j] < A[min]:

                min = j

        A[i], A[min] = A[min], A[i]

1. Demo

* Tạo mảng testArray tượng trưng cho input mảng A :
* elements = list()

    times = list()

    for i in range(1, 10):

        testArray = randint(0, 1000 \* i, 1000 \* i)

        start = time.process\_time()

        SelectionSort(testArray)

        end = time.process\_time()

        # print("Sorted list is ", a)

        print(len(testArray), "Elements Sorted by Selection in ", end - start)

        elements.append(len(testArray))

        times.append(end - start)

* Sau đó tạo phương thức vẽ đồ thị
* plt.xlabel('List Length')

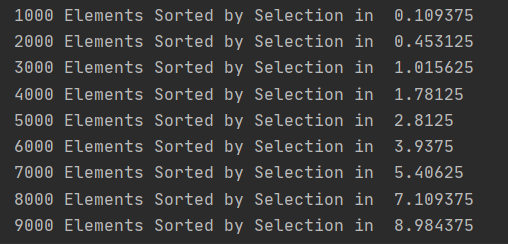
    plt.ylabel('Time Complexity')

    plt.plot(elements, times, label='Selection sort')

    plt.grid()

    plt.legend()

    plt.show()

* Sau đó thay hàm selection sort sẽ cho ta kết quả cụ thể về thời gian ở mảng 1000 đến 9000 : 
* Đây là biểu đồ :

Chart, line chart

Description automatically generated

* + 1. Buble sort

1. Ý tưởng

* Thuật toán so sánh các phần tử liền kề của danh sách và trao đổi chúng nếu chúng không theo thứ tự. Bằng cách lặp đi lặp lại, cuối cùng kết thúc bằng cách “sủi bọt” phần tử lớn nhất về vị trí cuối cùng của danh sách. Và lần lượt tiếp theo, tiếp đến các phần tử lớn thứ 2 cho đến khi lần thứ n – 1 và danh sách sẽ được sắp xếp.

1. Khai triển

#Sorts a given array by bubble sort

#Input: An array A[0..n − 1] of orderable elements

#Output: Array A[0..n − 1] sorted in nondecreasing order

def bubbleSort(A):

    n = len(A)

    for i in range(0, n-1):

        for j in range(0,n-1-i):

            if A[j+1] < A[j]:

                A[j], A[j+1] = A[j+1], A[j]

    return A

1. Demo

* Khởi tạo một mảng testArray để tượng cho mảng A nhập vào :
* elements = list()

    times = list()

    for i in range(1, 10):

        testArray = randint(0, 1000\*i, 1000\*i)

        start = time.clock()

        bubbleSort(testArray,0, len(testArray) -1)

        end = time.clock()

        # print("Sorted list is ", a)

        print(len(testArray), "Elements Sorted by Bubblesort in ", end-start)

        elements.append(len(testArray))

        times.append(end-start)

* Sau đó vẽ đồ thị
* plt.xlabel('List Length')

    plt.ylabel('Time Complexity')

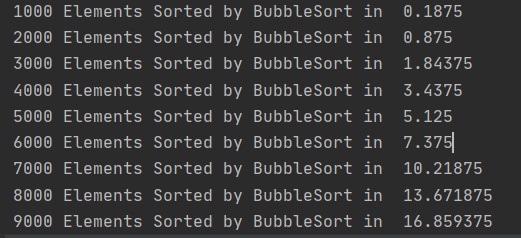
    plt.plot(elements, times, label ='Bubble sort')

    plt.grid()

    plt.legend()

    plt.show()

* Đây là kết quả thời gian chạy từ mảng 1000 đến 9000 :



* Đây là biểu đồ :

Chart, line chart

Description automatically generated

* + 1. Sequential search

1. Ý tưởng

* Thuật toán so sánh lần lượt các thành phần trong danh sách cho trước với khóa, thành phần cần tìm cho trước, cho đến khi có thành phần trùng với khóa ( tìm kiếm thành công ), hoặc không có thành phần nào trùng ( tìm kiếm thất bại ). Ở thuật toán tìm kiếm tuần tự này, chúng ta đính khóa vào cuối chuỗi các kí tự để loại trường hợp tìm kiếm cuối chuỗi.

1. Khai triển

#Input : Array A of n elements and variable x contain se rch key

#Output : The index of the first elements ; else -1 if not found

def sqtSearch(A,x):

    A[len(A)-1] = x

    i = 0

    while(A[i] != x):

        i +=1

    if (i < len(A)):

        return i

    else: return -1

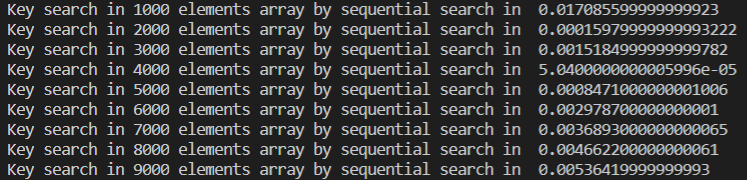
1. Demo

* Khởi tạo mảng testArray và khóa x ngẫu nhiên :

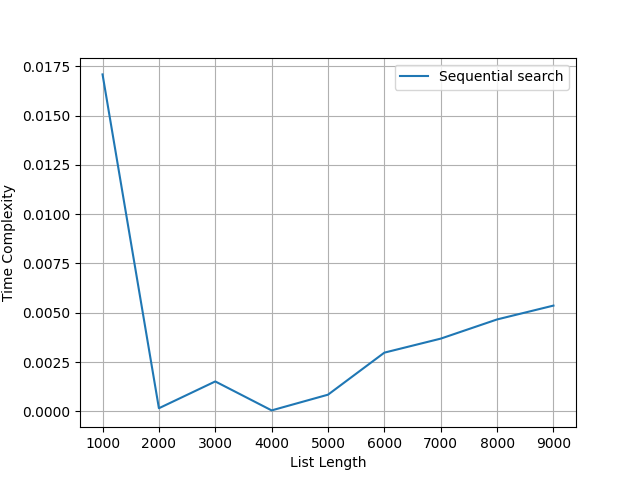
x = randint(0,1000)

for i in range(1, 10):

    testArray = randint(0, 1000\*i, 1000\*i)

* Thời gian chạy cho lần lượt các mảng từ 3000 đến 9000 kí tự:

- Sơ đồ thực thi theo thời gian :



### 2.2 Divide-and-conquer

A. Ý tưởng cơ bản :

- Vấn đề lớn được chia nhỏ thành các mảng vấn-đề-phụ ( lý tưởng là các mảng bằng nhau).

- Các vấn-đề-phụ được xử lí, thường thông qua đệ quy.

- Nếu cần thiết, giải pháp của các vấn-đề-phụ được hợp lại để lấy giải pháp cho vấn đề ban đầu.

B. Hệ thức truy hồi tổng quát của chia-để-trị :

T(n) = aT(n/b) + f(n)

Trong đó, n là độ dài input, b là số lần cần chia đôi chuỗi để giải bài toán và a là những phần cần được giải ( a >=1, b>1). Bên cạnh, hàm f(n) liên quan đến thời gian chia chuỗi có độ dài n thành các chuỗi con có độ dài n/b và hợp nhất các giải pháp lại.

#### 2.2.1 Mergesort

##### a. Ý tưởng

Sắp xếp thứ tự các phần tử trong mảng bằng cách lần lượt chia đôi mảng thành các cặp phần tử cho đến khi chỉ còn các phần tử đơn nhất và sau đó sắp xếp vị trí các phần tử một cách riêng biệt và ghép các cặp phần tử lại thành một mảng các phần tử đã được sắp xếp.

##### b. Khai triển

def mergesort(A):

    #Sorts array A[0..n − 1] by recursive mergesort

    #Input: An array A[0..n − 1] of orderable elements

    #Output: Array A[0..n − 1] sorted in nondecreasing order

    n = len(A)

    if n > 1:

        middle = n // 2

        B = A[:middle]

        C = A[middle:]

        mergesort(B)

        mergesort(C)

        merge(B,C,A)

def merge(B,C,A):

    #Merges two sorted arrays into one sorted array

    #Input: Arrays B[0..p − 1] and C[0..q − 1] both sorted

    #Output: Sorted array A[0..p + q − 1] of the elements of B and C

    i = 0

    j = 0

    k = 0

    p = len(B)

    q = len(C)

    sizeA = len(A)

    while i < p and j < q:

        if B[i] < C[j]:

            A[k] = B[i]

            i = i + 1

        else:

            A[k] = C[j]

            j = j + 1

        k = k + 1

    middle = sizeA//2

    if i == p:

        A[k:] = C[j:q]

    else:

        A[k:] = B[i:p]

##### c. Demo

elements = list()

times = list()

for i in range(1, 10):

    testArray = randint(0, 1000\*i, 1000\*i)

    start = time.clock()

    mergesort(testArray)

    end = time.clock()

    print(len(testArray), "Elements Sorted by MergeSort in ", end-start)

    elements.append(len(testArray))

    times.append(end-start)

plt.xlabel('List Length')

plt.ylabel('Time Complexity')

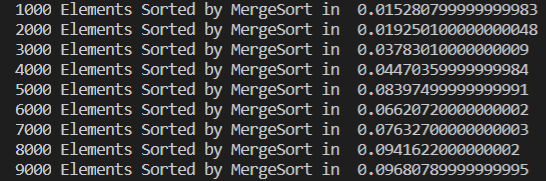
plt.plot(elements, times, label ='Merge sort')

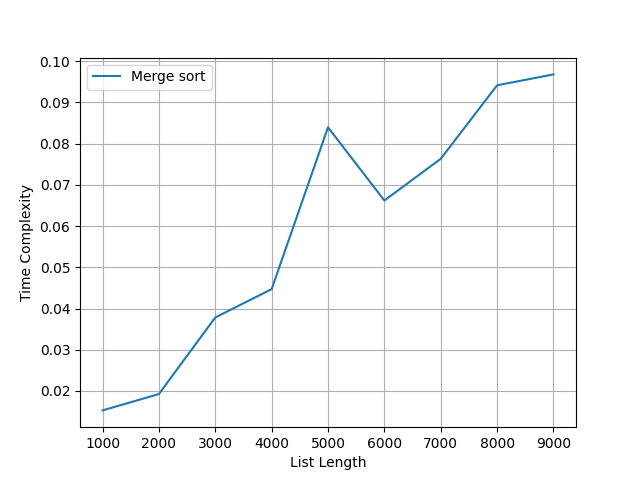
plt.grid()

plt.legend()

plt.show()

-Thời gian thực thi với input từ 1000 đến 9000 phần tử :



* Sơ đồ thực thi theo thời gian :

#### 2.2.2 Quicksort

##### a. Ý tưởng

- Chia mảng phần tử dựa trên giá trị của phần tử trục (pivot). Chọn một phần tử làm pivot và phân vùng mảng có sẵn theo pivot đó. Pivot có thể được chọn là phần tử đầu tiên, phần tử cuối cùng, một phần tử ngẫu nhiên trong mảng hoặc là phần tử trung vị. Thông thường, pivot là phần tử đầu tiên. Sau đó, khai báo 2 biến con trỏ để duyệt từ hai phía của phần tử trục. Biến con trỏ trái sẽ có nhiệm vụ tìm kiếm những phần tử lớn hơn phần tử trục và biến con trỏ phải sẽ tìm kiếm những phần tử bé hơn phần tử trục. Biến con trỏ trái sẽ duyệt từ bên trái của phần tử trục trong khi đó biến bên phải sẽ duyệt phần bên phải của phần tử trục.

##### b. Khai triển

#input : Array A[0,...n-1] with l and r are its left and right indices

#output : Partition of A[l..r] with the split position returned as

#purpose : This function takes the first element as pivot,  and places all smaller (smaller than pivot)

#to left of pivot and all greater elements to right

# of pivot

def partrition(A,l,r):

  p = A[l]

  i = l-1

  j = r+1

  while (i<j):

    i+=1

    while (A[i] < p):

      i = i + 1

    j-=1

    while (A[j] > p):

      j = j -1

    if (i >= j):

      return j

    A[i],A[j] = A[j],A[i]

  return j

#input : Array A[0,...n-1] with l and r are its left and right indices

#output : Array[l..r] sorted in nondecreasing orer

def quickSort(A,l,r):

  if l < r:

    s = partrition(A,l,r)

    quickSort(A,l,s)

    quickSort(A,s+1,r)

##### c. Demo

elements = list()

times = list()

for i in range(1, 10):

    testArray = randint(0, 1000\*i, 1000\*i)

    start = time.clock()

    quickSort(testArray,0, len(testArray) -1)

    end = time.clock()

    print(len(testArray), "Elements Sorted by QuickSort in ", end-start)

    elements.append(len(testArray))

    times.append(end-start)

plt.xlabel('List Length')

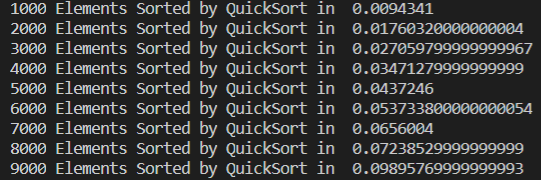
plt.ylabel('Time Complexity')

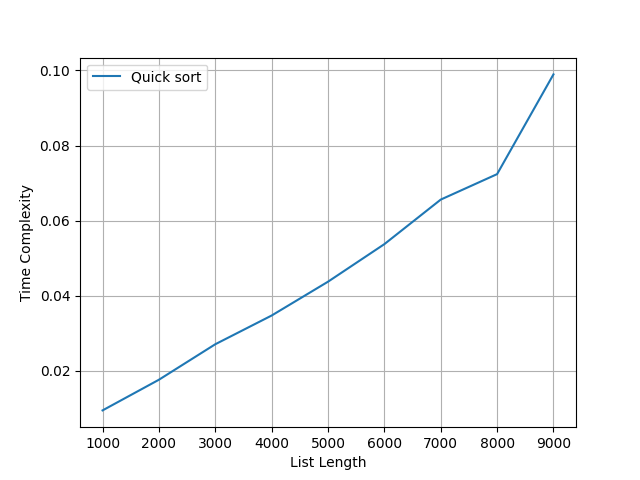
plt.plot(elements, times, label ='Quick sort')

plt.grid()

plt.legend()

plt.show()





2.3 Greedy algorithm

#### 2.4 Dynamic Programming

1. Ý tưởng cơ bản :

* Là một kĩ thuật thiết kế thuật toán theo kiểu chia bài toán lớn thành các bài toán con, sử dụng lời giải của các bài toán con để tìm lời giải cho bài toán ban đầu.

#### 2.4.1 Coin-row problem

1. Ý tưởng

* Cho một hàng gồm n đồng xu với giá trị nguyên dương không nhất thiết khác nhau. Mục đích là nhặt được số tiền tối đa với điều kiện là không nhặt 2 đồng tiền liền kề nhau.

1. Khai triển

# Three Basic Example - Algorithm 1

# Applies formula (8.3) bottom up to find the maximum amount of money that

# can be picked up from a coin row without picking two adjacent coins

# Input: Array C[1..n] of positive integers indicating the coin values

# Output: The maximum amount of money that can be picked up

def CoinRow(C):

    nC = len(C)

    nF = len(C) + 1

    F = [0] \* nF

    F[0] = 0

    F[1] = C[0]

    for i in range(2, nC + 1):

        F[i] = max(C[i - 1] + F[i - 2], F[i - 1])

    return F[nC]

1. Demo

* Tạo một mảng input array cho CoinRow() trong hàm main()

    elements = list()

    times = list()

    for i in range(35, 301):

        testArray = randint(1, 1000\*i, 1000 \* i)

        start = time.process\_time()

        CoinRow(testArray)

        end = time.process\_time()

        # print("Sorted list is ", a)

        print(len(testArray), "Elements Sorted by CoinRow in ", end - start)

        elements.append(len(testArray))

        times.append(end - start)

* Sau đó vẽ đồ thị

    plt.xlabel('List Length')

    plt.ylabel('Time Complexity')

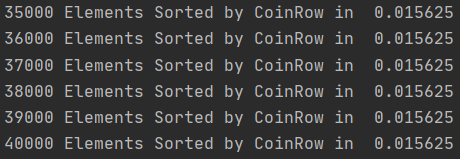
    plt.plot(elements, times, label='CoinRow')

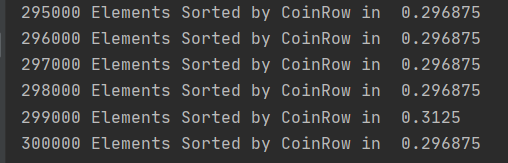
    plt.grid()

    plt.legend()

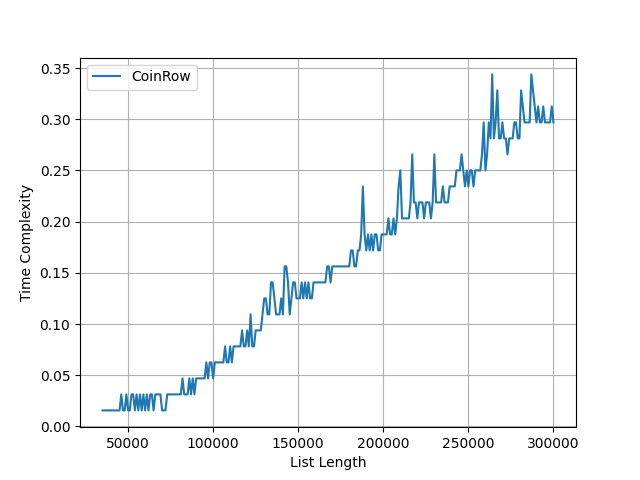
    plt.show()

* Thời gian chạy lần lượt cho mảng từ 35000 đến 299000 ký tự





* Sơ đồ thực thi theo thời gian :



2.4.2 Change-making problem

a. Ý tưởng

b. Khai triển

#Applies dynamic programming to find the minimum number of coins

#of denominations d1< d2 < . . . < dm where d1 = 1 that add up to a given amount n

#Input: Positive integer n and array D[1..m] of increasing positive integers indicating the coin denominations where D[1]= 1

#Output: The minimum number of coins that add up to n

def ChangeMaking(D,n):

    m = len(D)

    F = [0] \* (n+1)

    F[0] = 0

    for i in range (1, n+1):

        temp = math.inf

        j = 1

        while j <= m and i >= D[j-1]:

            temp = min(F[i-D[j-1]],temp)

            j = j + 1

        F[i] = temp + 1

    return F[n]

c. Demo

- Tạo một mảng input arrray cho hàm changeMaking

2.4.3 Knapsack

a. Ý tưởng

b. Khai triển

c. Demo

### 