| | |
|---|---|
| **Carátula para entrega de prácticas** | |
| Facultad de Ingeniería | Laboratorios de docencia |

*Profesor(a):* Jorge Alberto Solano Gálvez

*Asignatura:* Estructuras de Datos y Algoritmos II

*Grupo:* 2

*No de Práctica(s):* 3

*Integrante(s):* Valenzuela Ascencio Gustavo

*No. de Equipo de cómputo empleado:* 18

*Semestre:* 2024-1

*Fecha de entrega:* 8 de septiembre del 2023

*Observaciones:*

CALIFICACIÓN: ___________________

# Sorting Algorithms III.

**Objective:** Learn about the structure of sorting algorithms “Counting sort” and “Radix sort”.

**Activities:**

- Implement the Counting sort algorithm in Python language for sorting a data sequence.
- Implement the Radix sort algorithm in Python language for sorting a data sequence.

**Instructions:**

- Implement in Python the ascending-order sorting algorithms (Radix sort and Counting sort) for sorting nodes.
- Parting from the Python algorithms, obtain the polynomial of best, average and worst case of time complexity for Radix sort and Counting sort.
- Parting from the Python algorithms, generate the graph for different time instances (lists from 1 to 1000 elements for  the next cases).
    - Best case
    - Worst case
    - Average case

The practice must be done individually.

The practice is checked during the laboratory session and must be uploaded with all the code and the report in a compressed file, once qualified, to the SiCCAAD platform.

**Counting sort**

Is a sorting algorithm based on keys between a specific range. It works by counting the number of objects having distinct key values. Then do some arithmetic operations to calculate the position of each object in the output sequence.

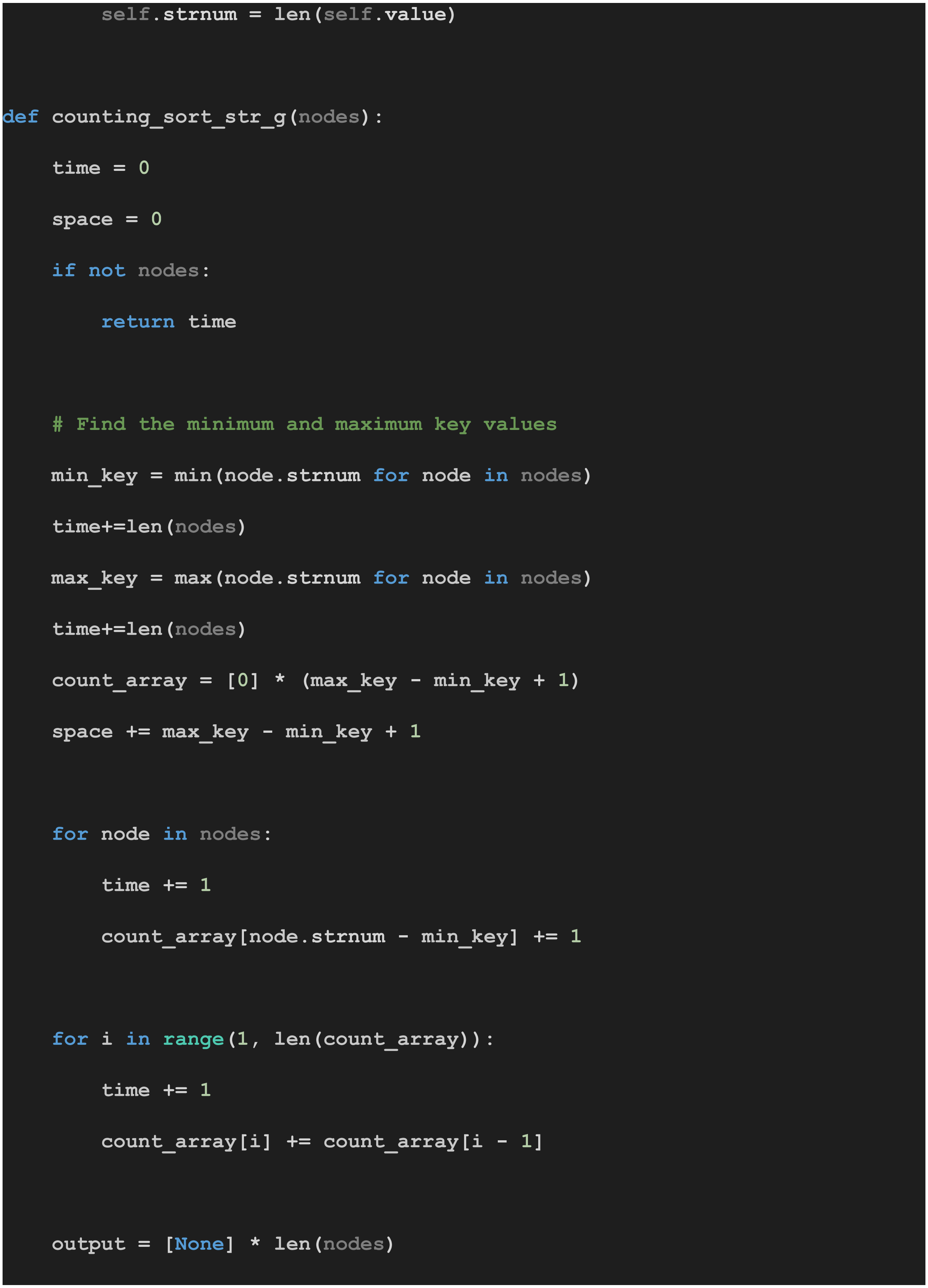
**Counting sort implementation and analysis using RAM model**

```python
#                                                TIME || SPACE

def counting_sort(arr):

    if len(arr) == 0:  # 5

        return arr # 1


    max_val = max(arr) # Max takes O(n) (Linear search) - 4n || 1

    min_val = min(arr) # Min takes O(n) (Linear search) - 4n || 1


    # Array of size m, being m = max_val


     count = [0] * (max_val - min_val + 1) # 6 | 6m (We are creating an array of size m, takes O(m) space)


    # arr size is n


    for num in arr: # 4(n+1)

        count[num - min_val] += 1 # 6n


    # Reconstruct the sorted array


    sorted_arr = [] # 4 | We build an array of size n, taking O(n) space


    for i in range(len(count)): #  5(m+1)
```

```
        sorted_arr.extend([i + min_val] * count[i]) # Append by the end in
the sorted array
                                                       # Also takes m time
because of the operation inside the parameter
                                                    # 5m


    return sorted_arr # 1


# TIME polynomial: 18n + 10m + 15 = O(n + m)

# SPACE polynomial: 2n + m + 1 = O(n + m)

# SPACE polynomial: 2n + m + 1 = O(n + m)


arr = [4, 2, 2, 8, 3, 3, 1, -5, -2]

sorted_arr = counting_sort(arr)

print("Original array:", arr)

print("Sorted array:", sorted_arr)
```

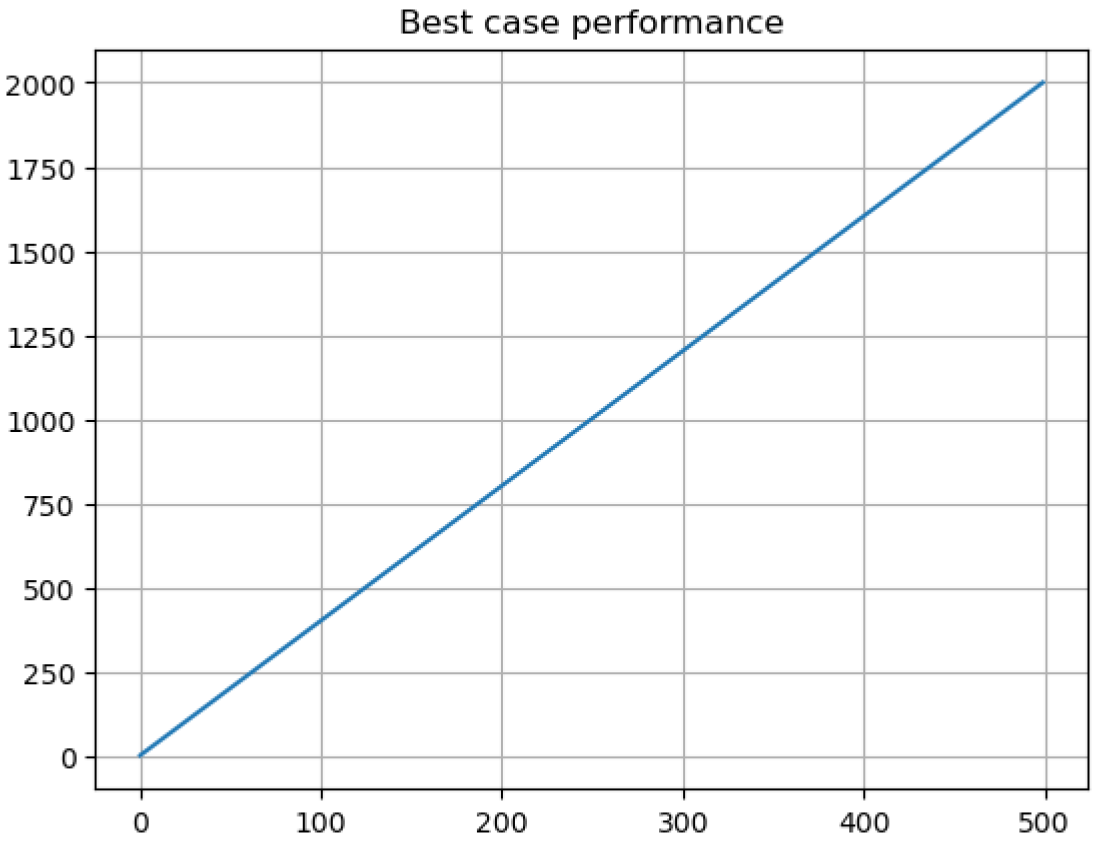
## Counting sort complexity graphs.

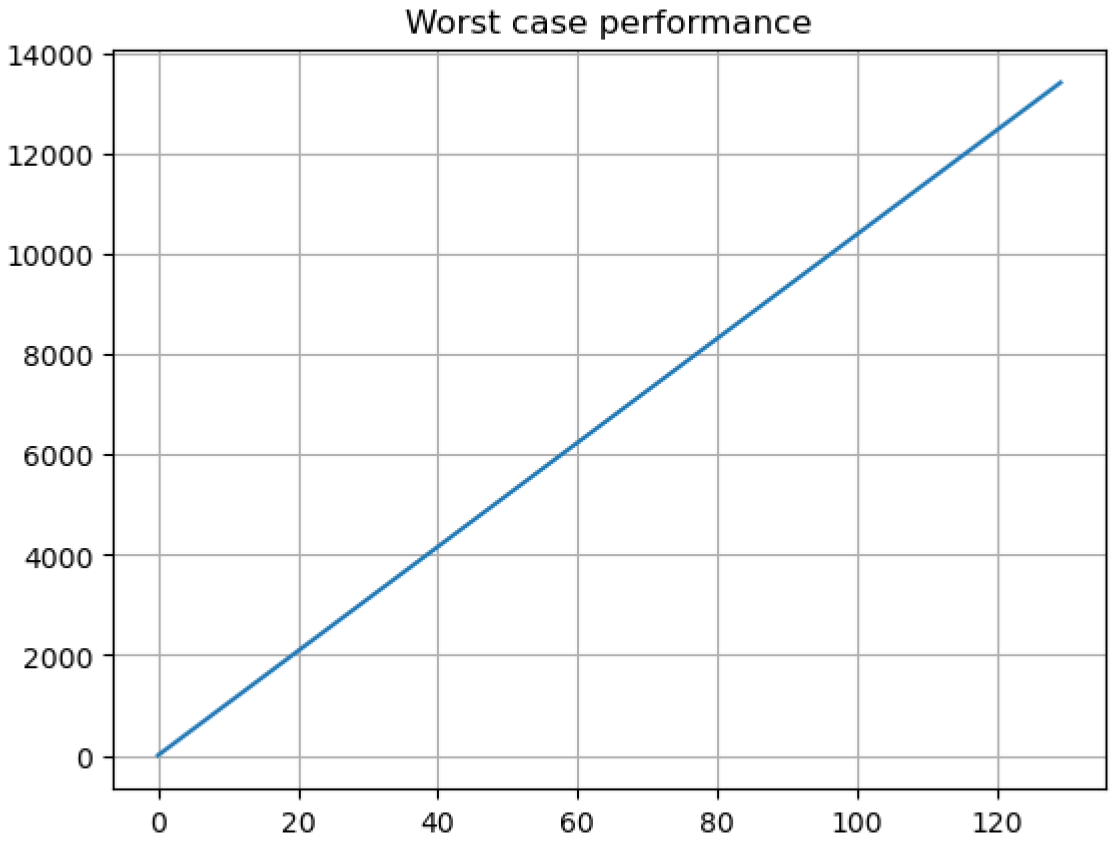
```
import csv

import matplotlib.pyplot as plt

class Node:

    def __init__(self,key,value):

        self.key = key

        self.value = value
```

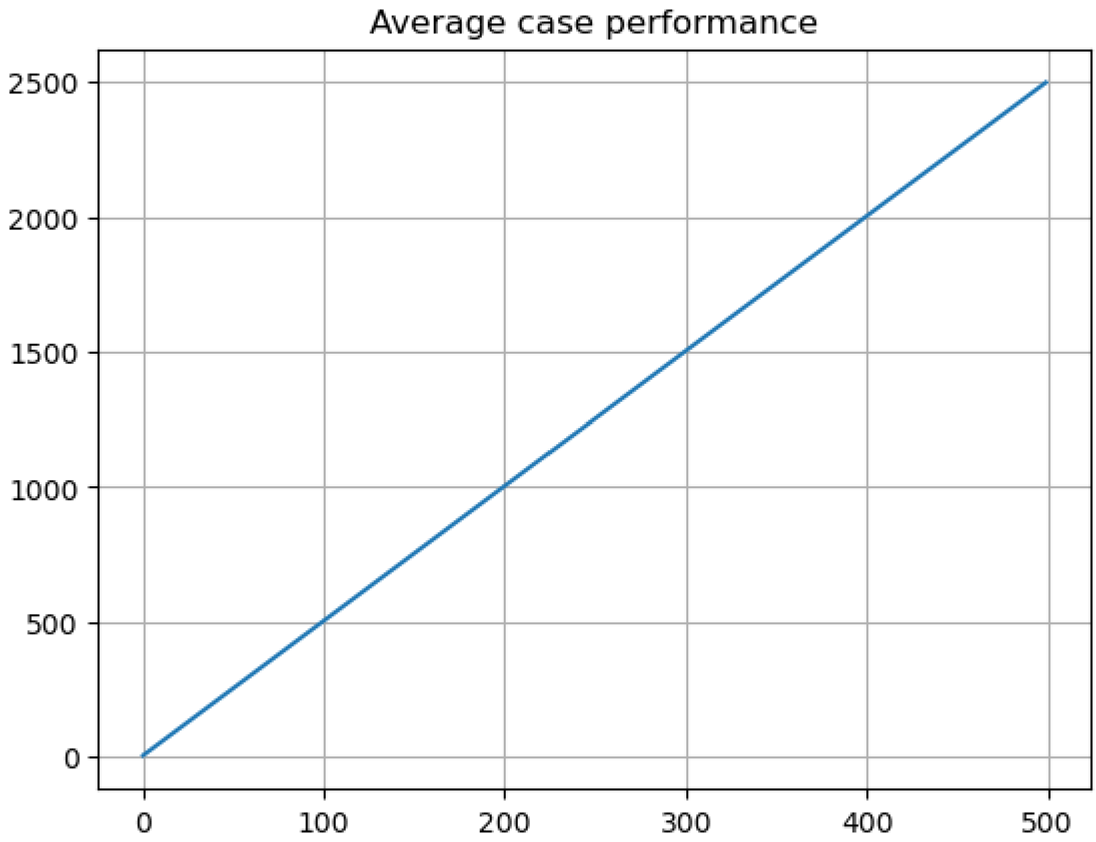


```
        self.strnum = len(self.value)

def counting_sort_str_g(nodes):
    time = 0
    space = 0
    if not nodes:
        return time

    # Find the minimum and maximum key values
    min_key = min(node.strnum for node in nodes)
    time+=len(nodes)
    max_key = max(node.strnum for node in nodes)
    time+=len(nodes)
    count_array = [0] * (max_key - min_key + 1)
    space += max_key - min_key + 1

    for node in nodes:
        time += 1
        count_array[node.strnum - min_key] += 1

    for i in range(1, len(count_array)):
        time += 1
        count_array[i] += count_array[i - 1]

    output = [None] * len(nodes)
```

```python
    for node in reversed(nodes):
        time += 1
        output[count_array[node.strnum - min_key] - 1] = node
        count_array[node.strnum - min_key] -= 1

    return time

def counting_sort_g(nodes):
    time = 0
    if not nodes:
        return []

    # Find the minimum and maximum key values
    min_key = min(node.key for node in nodes)
    time+=len(nodes)
    max_key = max(node.key for node in nodes)
    time+=len(nodes)

    count_array = [0] * (max_key - min_key + 1)

    for node in nodes:
        time+=1
        count_array[node.key - min_key] += 1
    for i in range(1, len(count_array)):
        time+=1
        count_array[i] += count_array[i - 1]

    output = [None] * len(nodes)

    for node in reversed(nodes):
        time+=1
        output[count_array[node.key - min_key] - 1] = node
        count_array[node.key- min_key] -= 1

    return time

x = []
y = []

for i in range(500):
    x.append(i)
    csv_file = 'bestCaseCounting2.csv'

    node_list = []

    # Open and read the CSV file
    with open(csv_file, mode='r', newline='') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
                # Assuming the first column is an integer and the second column is a string
            key = int(row[0])
            value = row[1]
            node = Node(key, value)
            node_list.append(node)
            tmp = counting_sort_str_g(node_list)
            if(len(y) < 500):
                y.append(tmp)

plt.plot(x,y)
plt.title('Best case performance')
plt.grid(True)
plt.show()


x = []
y = []

for i in range(130):
    x.append(i)
    csv_file = 'worstCaseCount.csv'

    node_list = []
    # Open and read the CSV file
    with open(csv_file, mode='r', newline='') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
                # Assuming the first column is an integer and the second column is a string
            key = int(row[0])
            value = row[1]
            node = Node(key, value)
            node_list.append(node)
            tmp = counting_sort_g(node_list)
            if(len(y) < 130):
                y.append(tmp)

plt.plot(x,y)
plt.title('Worst case performance')
plt.grid(True)
plt.show()


x = []
y = []

for i in range(500):
    x.append(i)
    csv_file = 'bestCaseCounting.csv'

    node_list = []

    # Open and read the CSV file
    with open(csv_file, mode='r', newline='') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
            # Assuming the first column is an integer and the second column is a string
            key = int(row[0])
            value = row[1]
            node = Node(key, value)
            node_list.append(node)
            tmp = counting_sort_g(node_list)
            if(len(y) < 500):
                y.append(tmp)

plt.plot(x,y)
plt.title('Average case performance')
plt.grid(True)
plt.show()
```


Best case performance
2000
1750
1500
1250
1000
750
500
250
0
0
100
200
300
400
500


Worst case performance
14000
12000
10000
8000
6000
4000
2000
0
0
20
40
60
80
100
120



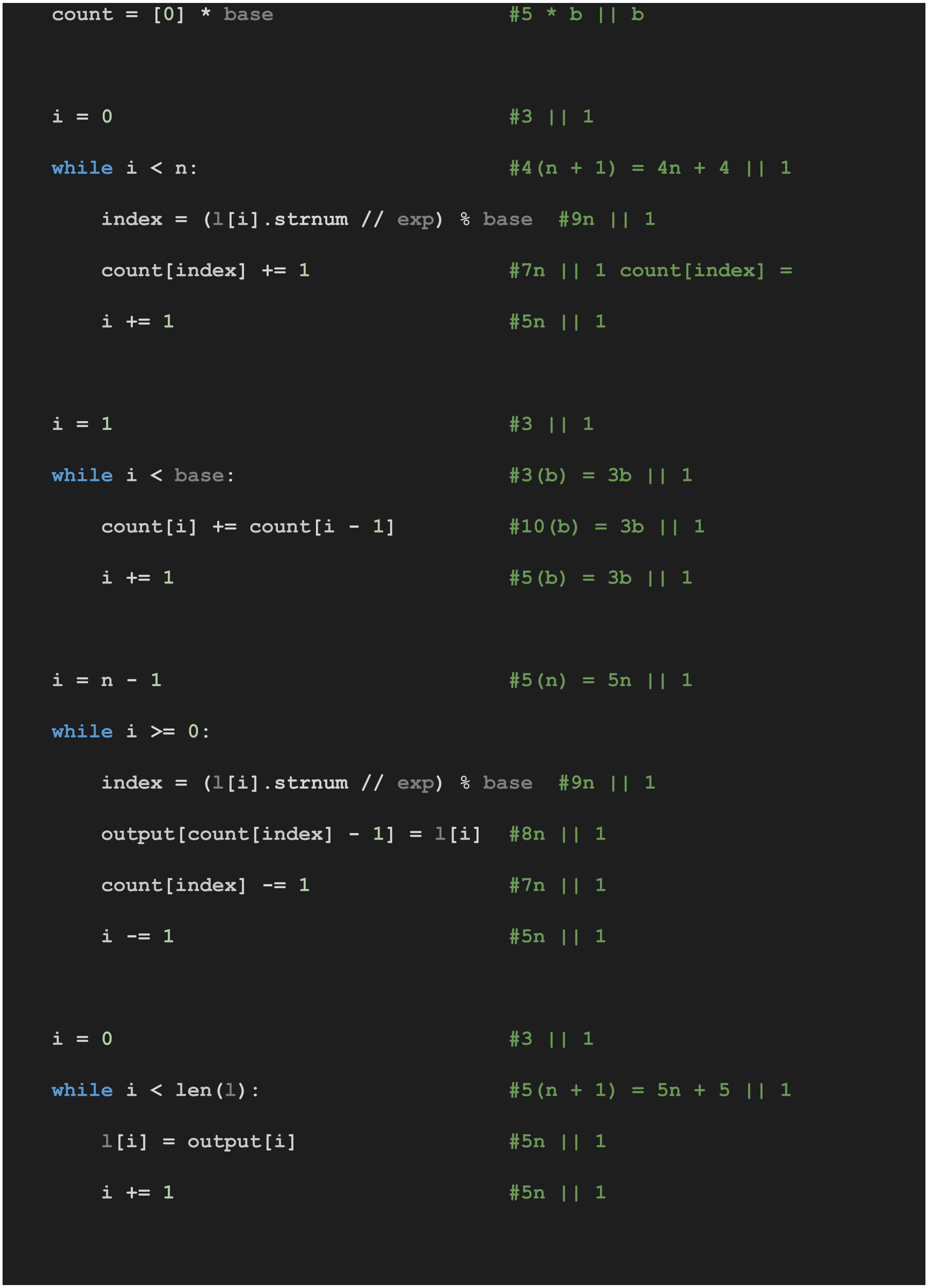
**Radix sort.**

Radix sort dates back as far as 1887 to the work of Herman Hollerith on tabulating machines. Radix sorting algorithms came into common use as a way to sort punched cards as early as 1923.

The first memory-efficient computer algorithm for this sorting method was developed in 1954 at MIT by Harold H. Seward.

**Implementation and analysis using RAM model.**

```
class Node:

    def __init__(self,key,value):

        self.key = key

        self.value = value
        self.strnum = len(self.value)


def radix_sort(l):
    max_id = max(node.key for node in l)  #n || k
    exp = 1                              #3 || 1
    base = 10                            #3 || 1
    digit = 0                            #3 || 1

    while digit < max_id:                #4k || 1
          bucket_sort(l, exp, base)         #k * (79n + 14b + 17) (b is a constant)
                                         # Complexity: O(n)


        exp *= 10                        #5k || 1
        digit += 1                       #5k || 1


def bucket_sort(l, exp, base):
    n = len(l)                           #4 ||
    output = [0] * n                     #5n || n
    count = [0] * base                   #5 * b || b

    i = 0                                #3 || 1
    while i < n:                         #4(n + 1) = 4n + 4 || 1
        index = (l[i].key // exp) % base  #9n || 1
        count[index] += 1                #7n || 1 count[index] =
        i += 1                           #5n || 1

    i = 1                                #3 || 1
    while i < base:                      #3(b) = 3b || 1
        count[i] += count[i - 1]         #10(b) = 3b || 1
        i += 1                           #5(b) = 3b || 1

    i = n - 1                            #5(n) = 5n || 1
    while i >= 0:
        index = (l[i].key // exp) % base  #9n || 1
        output[count[index] - 1] = l[i]  #8n || 1
        count[index] -= 1                #7n || 1
        i -= 1                           #5n || 1

    i = 0                                #3 || 1
    while i < len(l):                    #5(n + 1) = 5n + 5 || 1
        l[i] = output[i]                 #5n || 1
        i += 1                           #5n || 1

# Time complexity (Bucket sort) = 79n + +14b 17
# Complexity: O(n)

# SPACE (Bucket sort)
# n + b + 17
# Complexity: O(n)
# Time complexity (Radix sort) = k(79n + 14b + 17) + 14k + n + 9 = 79nk + 14kb + 31k + 9

# Complexity = O(nk)


# SPACE (Radix sort)

# (n + b + 17) + k + 6 = n + b + + k 24

# Complexity: O(n + k)


def radix_sort_len(l):

    max_id = max(node.strnum for node in l)  #n || k

    exp = 1                              #3 || 1

    base = 10                            #3 || 1

    digit = 0                            #3 || 1


    while digit < max_id:                #4k || 1

        bucket_sort_len(l, exp, base)         #k * (79n + 14b + 17) (b is a constant)

                                         # Complexity: O(n)


        exp *= 10                        #5k || 1

        digit += 1                       #5k || 1


def bucket_sort_len(l, exp, base):

    n = len(l)                           #4 ||

    output = [0] * n                     #5n || n
```



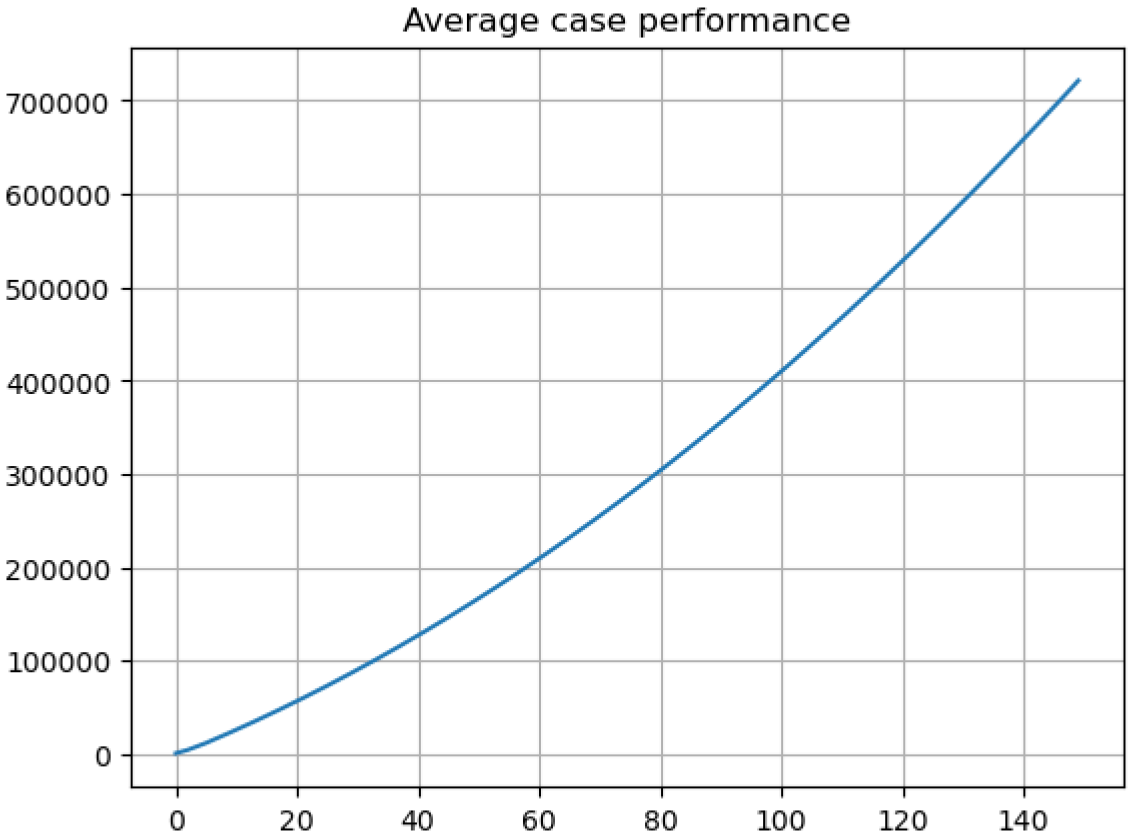
```
count = [0] * base                     #5 * b || b


i = 0                                  #3 || 1

while i < n:                           #4(n + 1) = 4n + 4 || 1

    index = (l[i].strnum // exp) % base  #9n || 1

    count[index] += 1                  #7n || 1 count[index] =

    i += 1                             #5n || 1


i = 1                                  #3 || 1

while i < base:                        #3(b) = 3b || 1

    count[i] += count[i - 1]           #10(b) = 3b || 1

    i += 1                             #5(b) = 3b || 1


i = n - 1                              #5(n) = 5n || 1

while i >= 0:

    index = (l[i].strnum // exp) % base  #9n || 1

    output[count[index] - 1] = l[i]    #8n || 1

    count[index] -= 1                  #7n || 1

    i -= 1                             #5n || 1


i = 0                                  #3 || 1

while i < len(l):                      #5(n + 1) = 5n + 5 || 1

    l[i] = output[i]                   #5n || 1

    i += 1                             #5n || 1
```

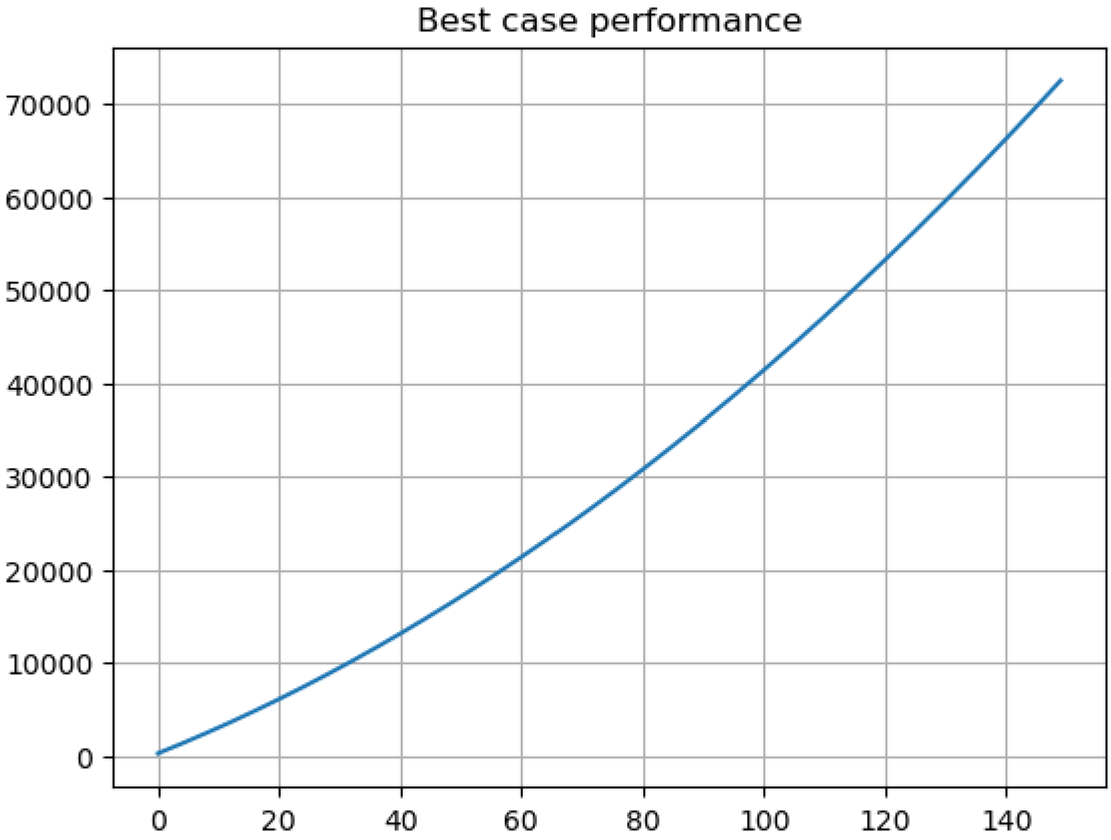
```
def radix_sort_s(nodes):
    max_length = max(len(node.value) for node in nodes)
    base = 256

    for digit in range(max_length - 1, -1, -1):
        bucket_sort_s(nodes, digit, base)

def bucket_sort_s(nodes, digit, base):
    n = len(nodes)
    output = [None] * n
    count = [0] * base

    for i in range(n):
        if digit < len(nodes[i].value):
            index = ord(nodes[i].value[digit])
        else:
            index = 0
        count[index] += 1

    for i in range(1, base):
        count[i] += count[i - 1]

    for i in range(n - 1, -1, -1):
        if digit < len(nodes[i].value):
            index = ord(nodes[i].value[digit])

        else:

            index = 0

        output[count[index] - 1] = nodes[i]

        count[index] -= 1


    for i in range(n):

        nodes[i] = output[i]


nodes = []


tmp1 = Node(9,"Mafer Ayala")

tmp2 = Node(0,"Yordi Josue")

tmp3 = Node(6,"Quique")

tmp4 = Node(7,"Jans")

tmp5 = Node(5,"Aiti")

tmp6 = Node(10,"Gus")

tmp7 = Node(70,"Saul")

tmp8 = Node(4,"Arnau")

tmp9 = Node(17,"Emilio")

tmp10 = Node(80,"Kaz")


nodes.append(tmp1)

nodes.append(tmp2)

nodes.append(tmp3)
nodes.append(tmp4)

nodes.append(tmp5)

nodes.append(tmp6)

nodes.append(tmp7)

nodes.append(tmp8)

nodes.append(tmp9)

nodes.append(tmp10)




print('------------------------------BY
ID-------------------------------')


radix_sort(nodes)

for node in nodes:

    print(f'Key: {node.key}, Value: {node.value}')


print('------------------------------BY
LENGTH-------------------------------')


radix_sort_len(nodes)

for node in nodes:

    print(f'Key: {node.key}, Value: {node.value}')




print('------------------------------ LEXICOGRAPHICALLY
-------------------------------')
```

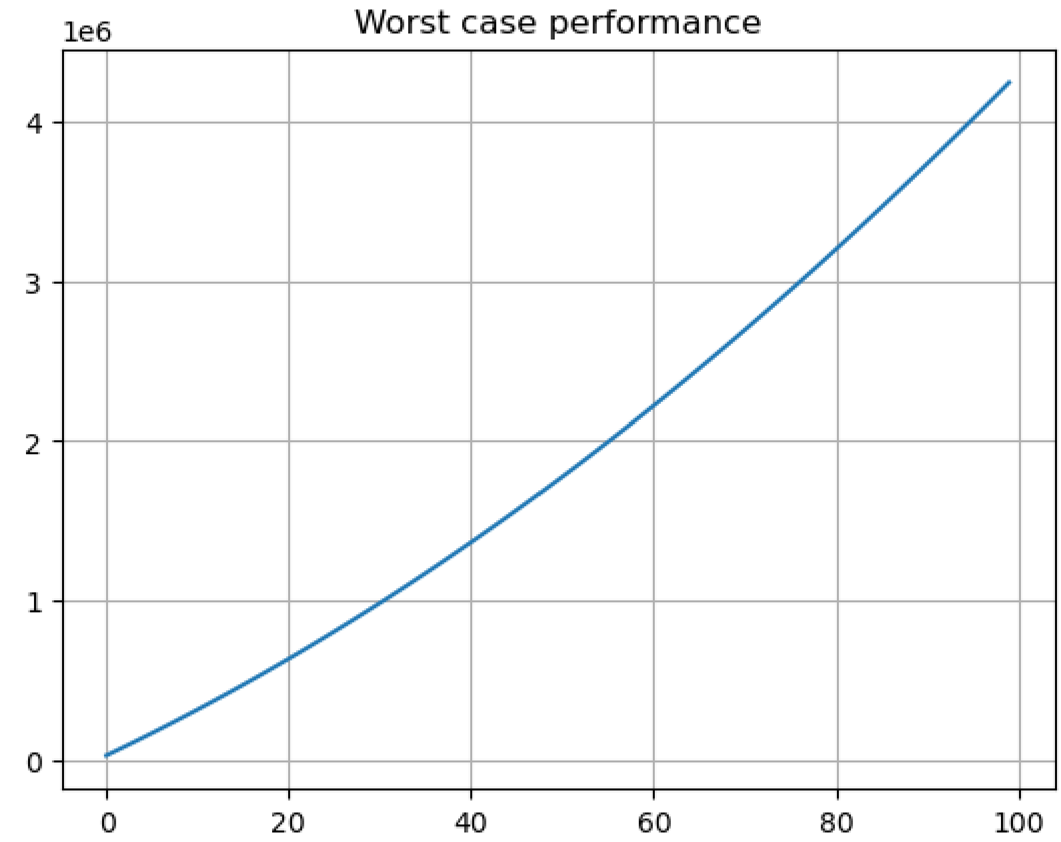
```
radix_sort_s(nodes)
for node in nodes:
    print(f'Key: {node.key}, Value: {node.value}')
```

## Complexity cases graphs.

```
import csv
import matplotlib.pyplot as plt

time = 0
def radix_sort_s_g(nodes):
    global time
    max_length = max(len(node.value) for node in nodes)
    base = 256

    for digit in range(max_length - 1, -1, -1):
        time += 1
        bucket_sort_s_g(nodes, digit, base)

def bucket_sort_s_g(nodes, digit, base):
    global time
    time+=1
    n = len(nodes)
    output = [None] * n
    count = [0] * base

    for i in range(n):
        time+=1
        if digit < len(nodes[i].value):
            index = ord(nodes[i].value[digit])
        else:
            index = 0
        count[index] += 1

    for i in range(1, base):
        time+=1
        count[i] += count[i - 1]

    for i in range(n - 1, -1, -1):
        time+=1
        if digit < len(nodes[i].value):
            index = ord(nodes[i].value[digit])
        else:
            index = 0
        output[count[index] - 1] = nodes[i]
        count[index] -= 1
    for i in range(n):
        time+=1
        nodes[i] = output[i]

x = []
y = []

for i in range(150):
    x.append(i)
    csv_file = 'averageCaseRadix.csv'

    node_list = []

    # Open and read the CSV file
    with open(csv_file, mode='r', newline='') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
                # Assuming the first column is an integer and the second column is a string
            key = int(row[0])
            value = row[1]
            node = Node(key, value)
            node_list.append(node)
            tmp = radix_sort_s_g(node_list)
            if(len(y) < 150):
                y.append(time)
        time = 0

plt.plot(x,y)
plt.title('Average case performance')
plt.grid(True)
plt.show()

x=[]
y=[]

for i in range(150):
    x.append(i)
    csv_file = 'bestCaseRadix.csv'

    node_list = []

    # Open and read the CSV file
    with open(csv_file, mode='r', newline='') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
                # Assuming the first column is an integer and the second column is a string
            key = int(row[0])
            value = row[1]
            node = Node(key, value)
            node_list.append(node)
            tmp = radix_sort_s_g(node_list)
            if(len(y) < 150):
                y.append(time)
        time = 0

plt.plot(x,y)
plt.title('Best case performance')
plt.grid(True)
plt.show()

x=[]
y=[]

for i in range(100):
    x.append(i)
    csv_file = 'worstRadix.csv'

    node_list = []

    # Open and read the CSV file
    with open(csv_file, mode='r', newline='') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
                # Assuming the first column is an integer and the second column is a string

            key = int(row[0])
            value = row[1]
            node = Node(key, value)
            node_list.append(node)
            tmp = radix_sort_s_g(node_list)
            if(len(y) < 100):
                y.append(time)
        time = 0

plt.plot(x,y)
plt.title('Worst case performance')
plt.grid(True)
plt.show()
```


Average case performance
700000
600000
500000
400000
300000
200000
100000
0
0
20
40
60
80
100
120
140


Best case performance
70000
60000
50000
40000
30000
20000
10000
0
0
20
40
60
80
100
120
140



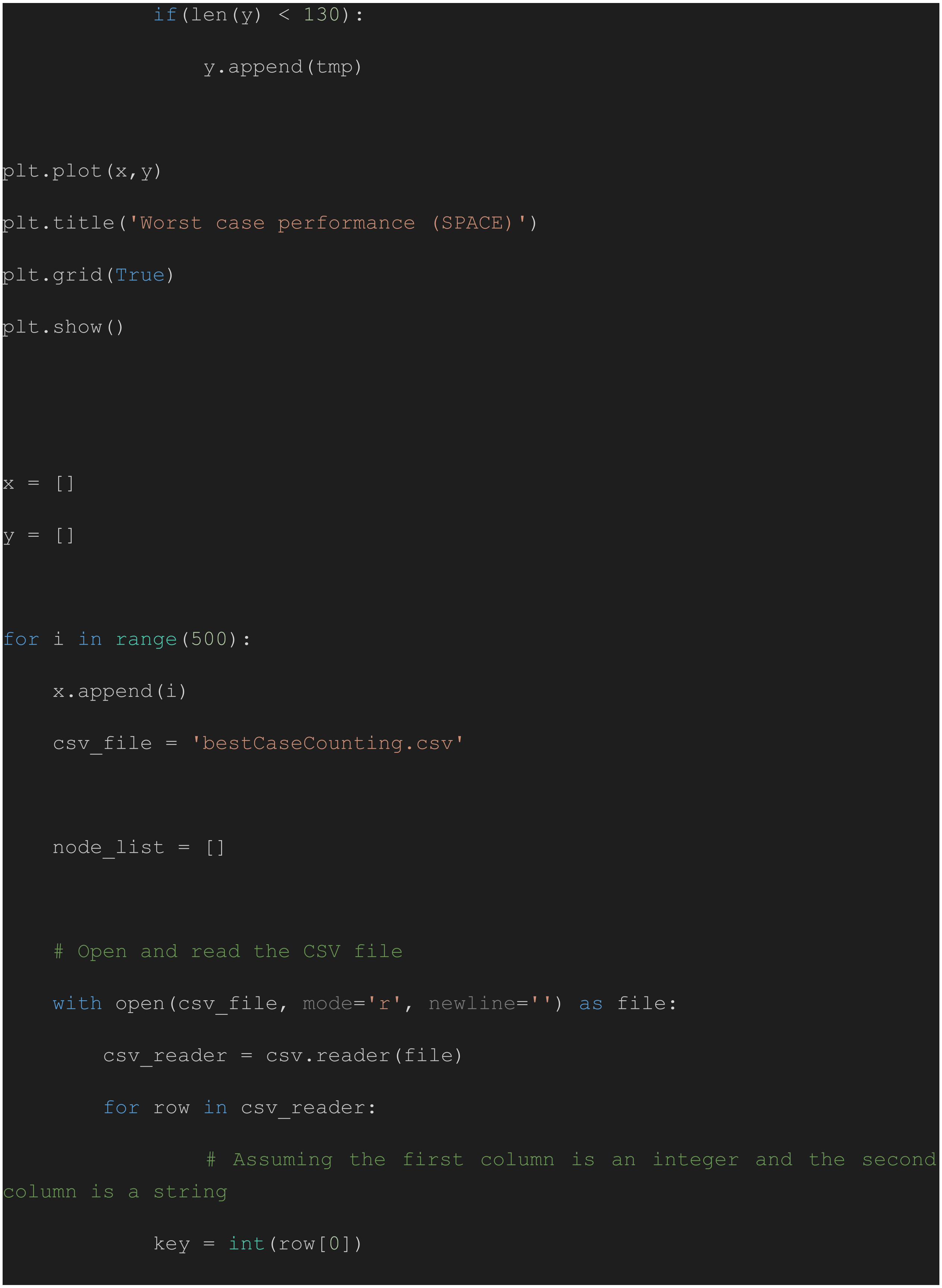
## Space complexity graphs.

## Counting sort.

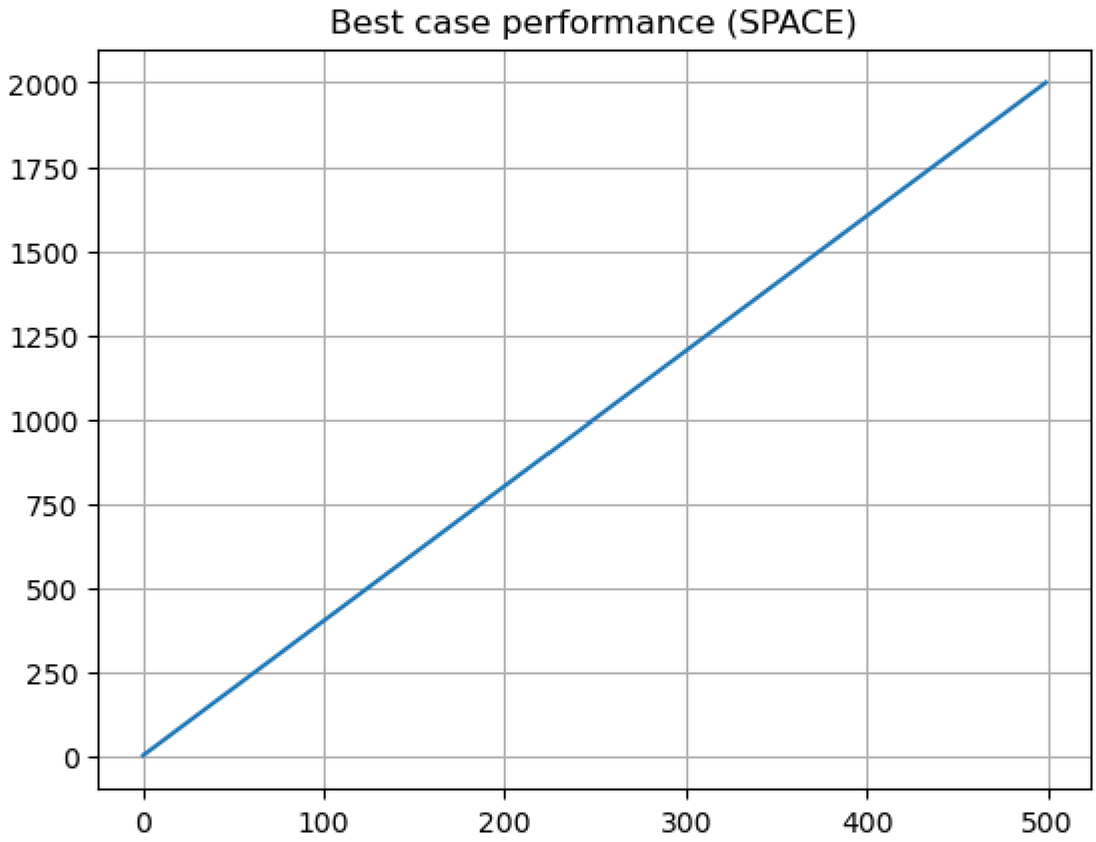
```
import csv
import matplotlib.pyplot as plt

class Node:
    def __init__(self,key,value):
        self.key = key
        self.value = value
        self.strnum = len(self.value)

def counting_sort_space(nodes):
```

```python
space = 0
space += len(nodes)
if not nodes:
    return []

# Find the minimum and maximum key values
min_key = min(node.key for node in nodes)
max_key = max(node.key for node in nodes)

count_array = [0] * (max_key - min_key + 1)
space += (max_key - min_key + 1)

for node in nodes:
    count_array[node.key - min_key] += 1

for i in range(1, len(count_array)):
    count_array[i] += count_array[i - 1]

output = [None] * len(nodes)

for node in reversed(nodes):
    output[count_array[node.key - min_key] - 1] = node
    count_array[node.key- min_key] -= 1

return space
```

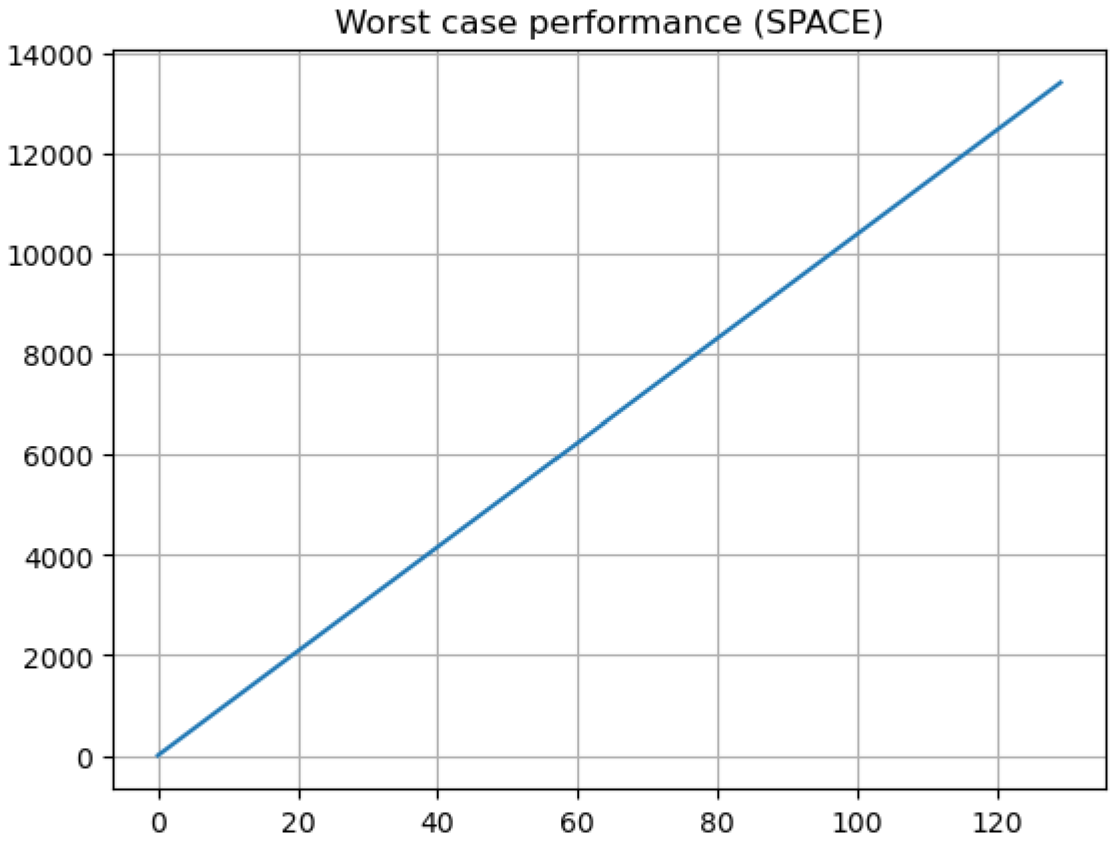
```
x = []
y = []

for i in range(500):
    x.append(i)
    csv_file = 'bestCaseCounting2.csv'

    node_list = []

    # Open and read the CSV file
    with open(csv_file, mode='r', newline='') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
            # Assuming the first column is an integer and the second column is a string
            key = int(row[0])
            value = row[1]
            node = Node(key, value)
            node_list.append(node)
            tmp = counting_sort_str_g(node_list)
            if(len(y) < 500):
                y.append(tmp)

plt.plot(x,y)
plt.title('Best case performance (SPACE)')

plt.grid(True)

plt.show()



x = []

y = []


for i in range(130):

    x.append(i)

    csv_file = 'worstCaseCount.csv'


    node_list = []


    # Open and read the CSV file

    with open(csv_file, mode='r', newline='') as file:

        csv_reader = csv.reader(file)

        for row in csv_reader:

                # Assuming the first column is an integer and the second column is a string

            key = int(row[0])

            value = row[1]

            node = Node(key, value)

            node_list.append(node)

            tmp = counting_sort_g(node_list)
```

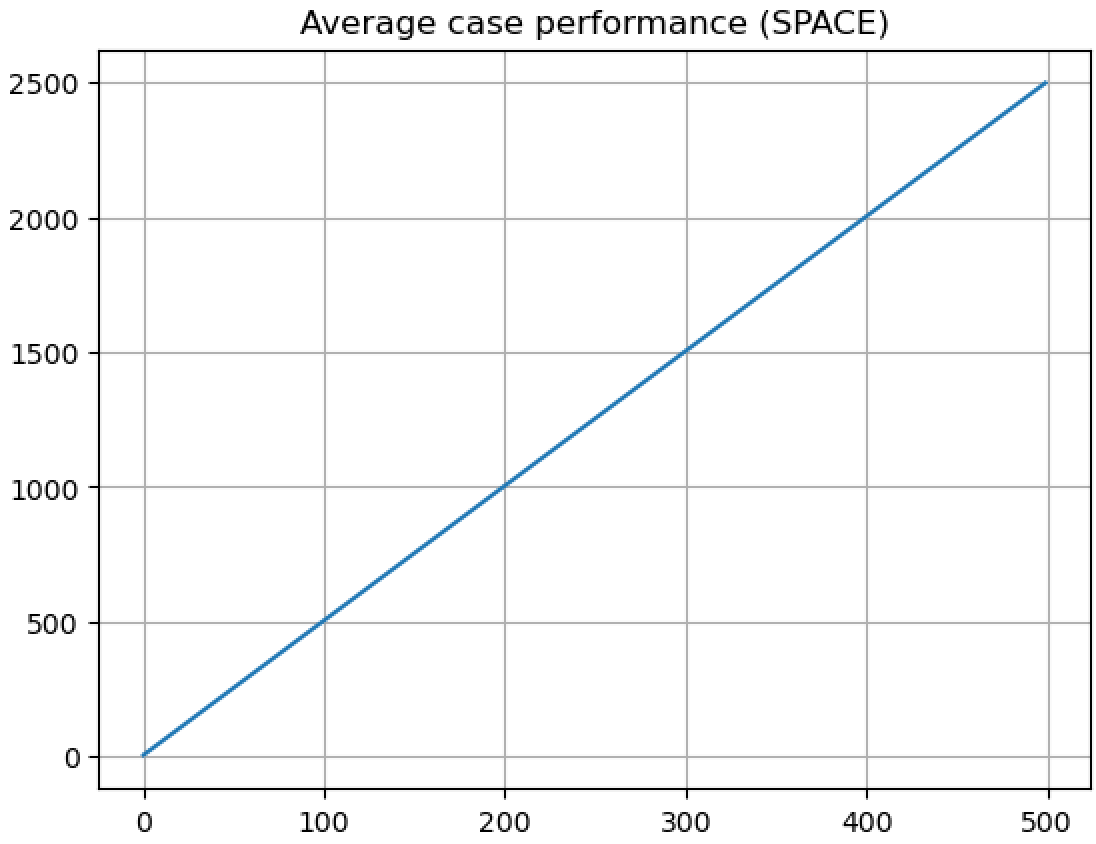


```
            if(len(y) < 130):
                y.append(tmp)

plt.plot(x,y)
plt.title('Worst case performance (SPACE)')
plt.grid(True)
plt.show()


x = []
y = []

for i in range(500):
    x.append(i)
    csv_file = 'bestCaseCounting.csv'

    node_list = []

    # Open and read the CSV file
    with open(csv_file, mode='r', newline='') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
                # Assuming the first column is an integer and the second column is a string
            key = int(row[0])
            value = row[1]
            node = Node(key, value)
            node_list.append(node)
            tmp = counting_sort_g(node_list)
            if(len(y) < 500):
                y.append(tmp)


plt.plot(x,y)
plt.title('Average case performance (SPACE)')
plt.grid(True)
plt.show()
```



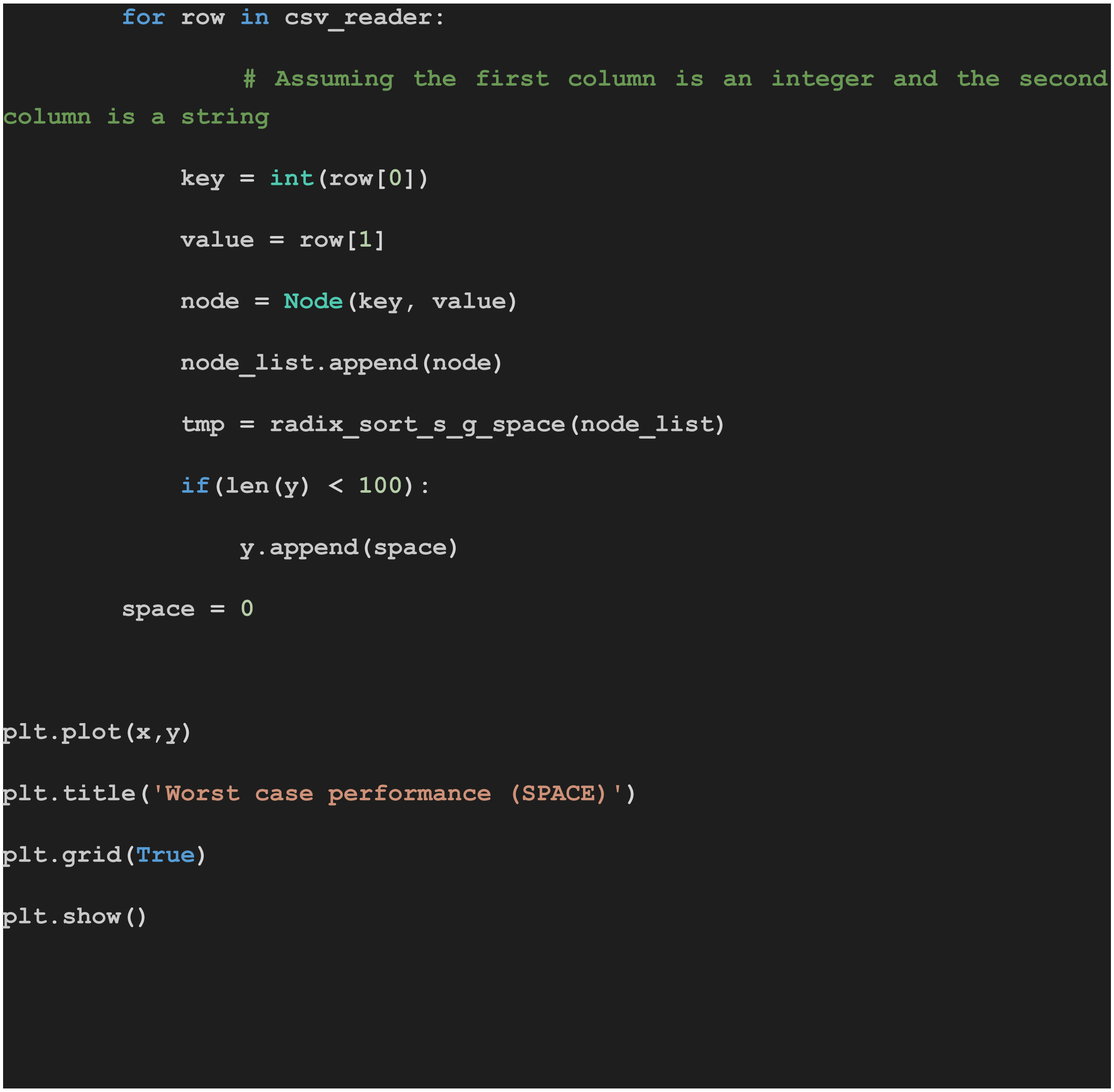
Best case performance (SPACE)


Worst case performance (SPACE)
14000
12000
10000
8000
6000
4000
2000
0
0
20
40
60
80
100
120

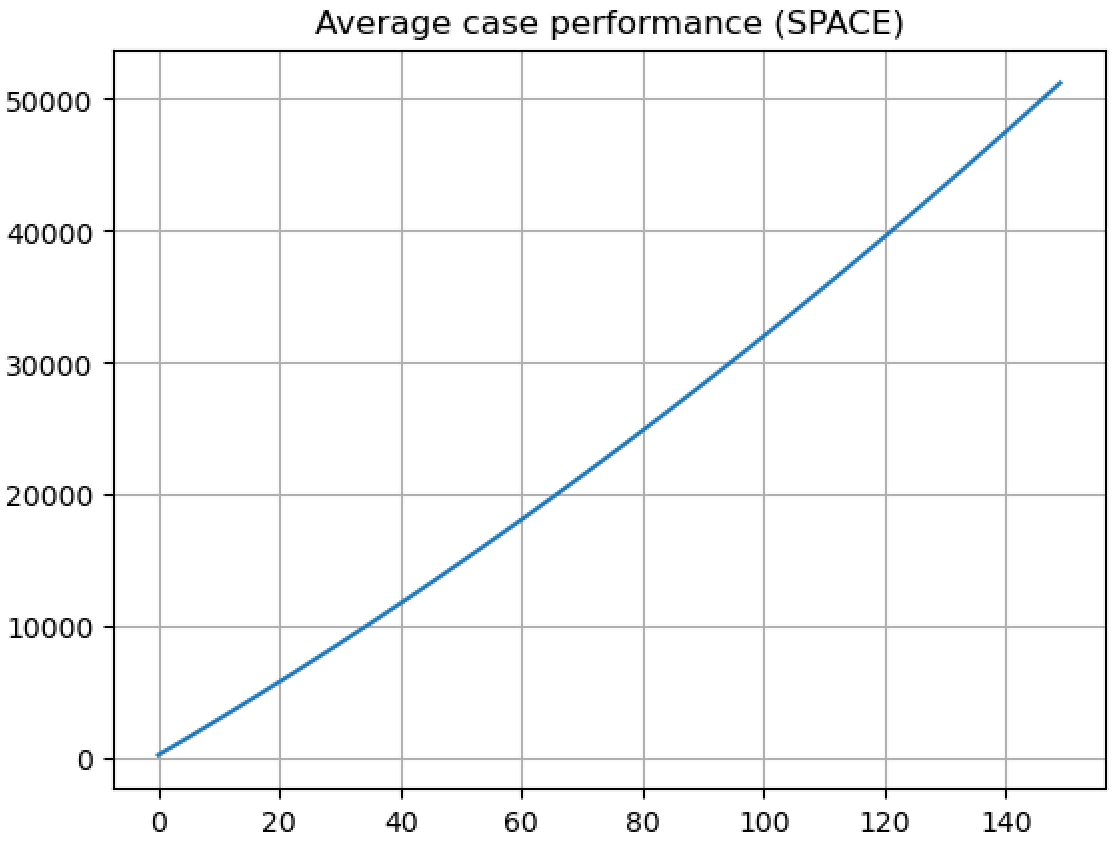


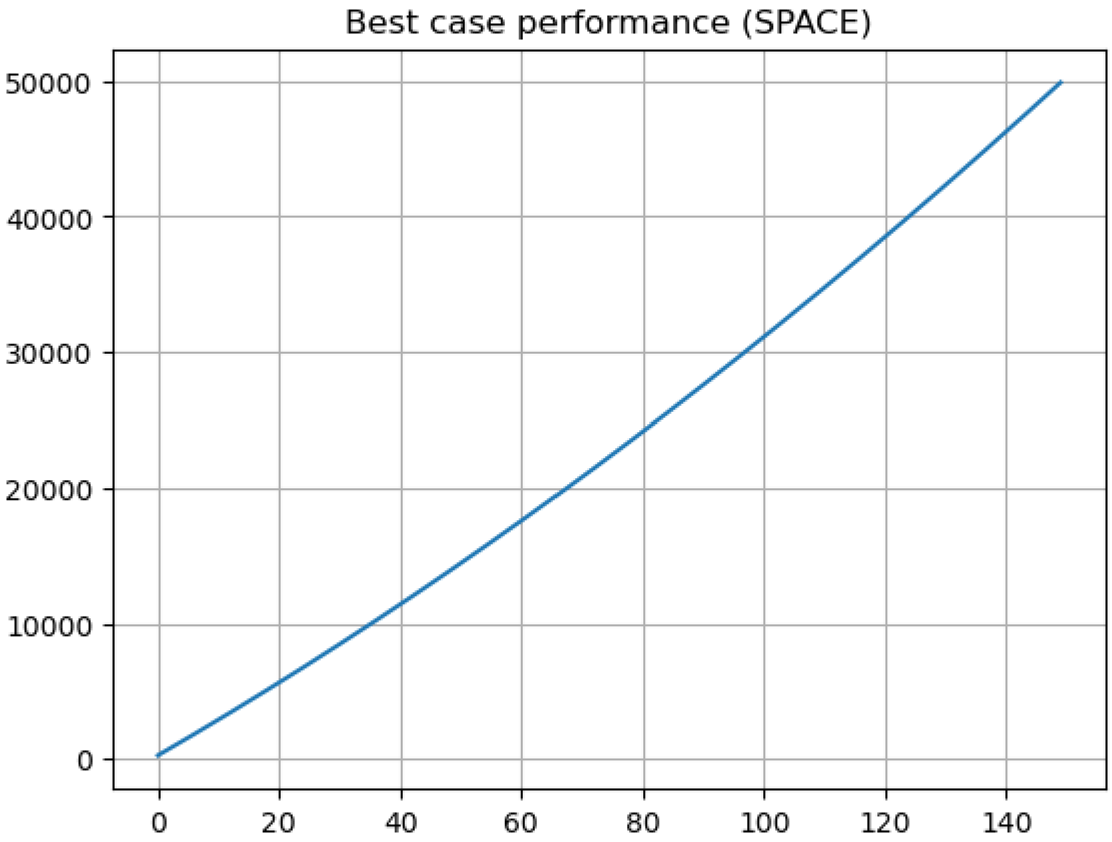
## Radix sort graph.

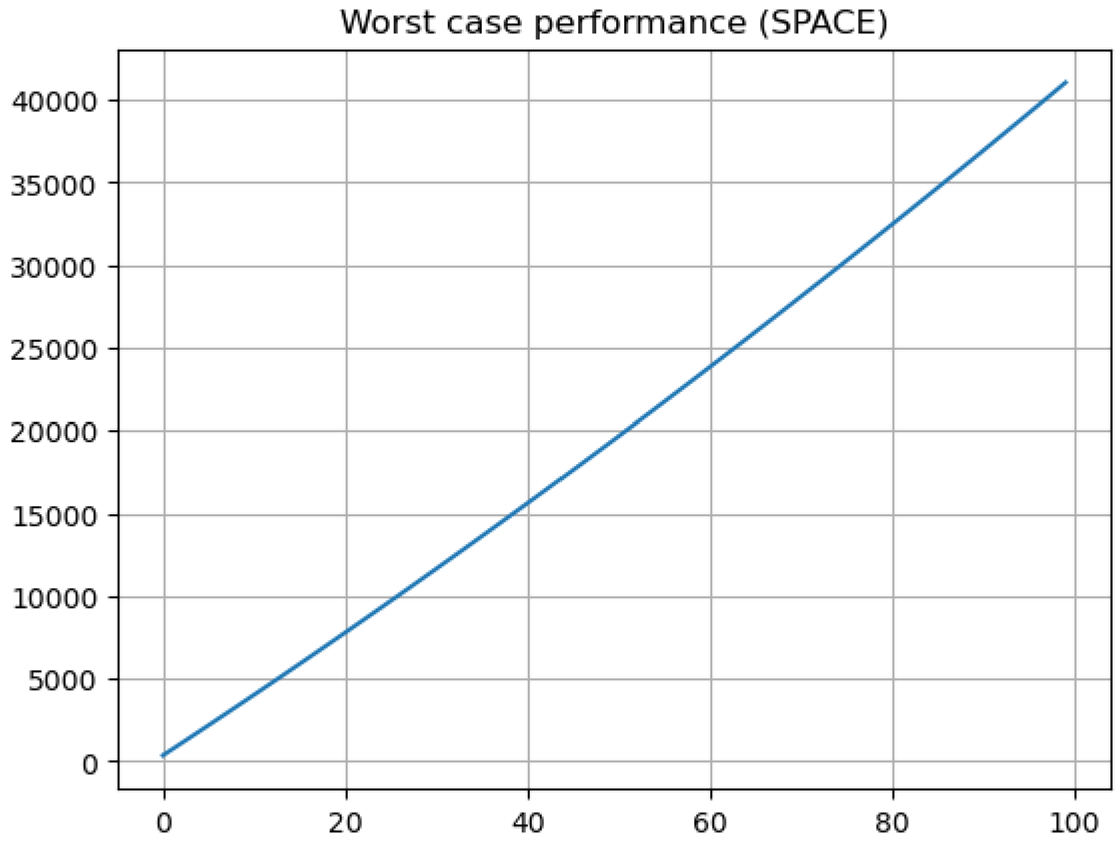
```
import csv
import matplotlib.pyplot as plt

space = 0
def radix_sort_s_g_space(nodes):
    global space
    max_length = max(len(node.value) for node in nodes)
    base = 256

    space += len(nodes)
    space += max_length
    space += base

    for digit in range(max_length - 1, -1, -1):
        bucket_sort_s_g(nodes, digit, base)

def bucket_sort_s_g(nodes, digit, base):
    n = len(nodes)
    output = [None] * n
    count = [0] * base

    for i in range(n):
        if digit < len(nodes[i].value):
            index = ord(nodes[i].value[digit])
        else:
            index = 0
        count[index] += 1

    for i in range(1, base):
        count[i] += count[i - 1]

    for i in range(n - 1, -1, -1):
        if digit < len(nodes[i].value):
            index = ord(nodes[i].value[digit])
        else:
            index = 0
        output[count[index] - 1] = nodes[i]
        count[index] -= 1
    for i in range(n):
        nodes[i] = output[i]

x = []
y = []

for i in range(150):
    x.append(i)
    csv_file = 'averageCaseRadix.csv'

    node_list = []

    # Open and read the CSV file
    with open(csv_file, mode='r', newline='') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
                # Assuming the first column is an integer and the second column is a string
            key = int(row[0])
            value = row[1]
            node = Node(key, value)
            node_list.append(node)
            tmp = radix_sort_s_g_space(node_list)
            if(len(y) < 150):
                y.append(space)
        space = 0

plt.plot(x,y)
plt.title('Average case performance (SPACE)')
plt.grid(True)
plt.show()

x=[]
y=[]

for i in range(150):
    x.append(i)
    csv_file = 'bestCaseRadix.csv'

    node_list = []

    # Open and read the CSV file
    with open(csv_file, mode='r', newline='') as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
                # Assuming the first column is an integer and the second column is a string
            key = int(row[0])
            value = row[1]

            node = Node(key, value)

            node_list.append(node)

            tmp = radix_sort_s_g_space(node_list)

            if(len(y) < 150):

                y.append(space)

        space = 0


plt.plot(x,y)

plt.title('Best case performance (SPACE)')

plt.grid(True)

plt.show()


x=[]

y=[]


for i in range(100):

    x.append(i)

    csv_file = 'worstRadix.csv'


    node_list = []


    # Open and read the CSV file

    with open(csv_file, mode='r', newline='') as file:

        csv_reader = csv.reader(file)
```



```
        for row in csv_reader:
                # Assuming the first column is an integer and the second column is a string
            key = int(row[0])
            value = row[1]
            node = Node(key, value)
            node_list.append(node)
            tmp = radix_sort_s_g_space(node_list)
            if(len(y) < 100):
                y.append(space)
        space = 0


plt.plot(x,y)
plt.title('Worst case performance (SPACE)')
plt.grid(True)
plt.show()
```


Average case performance (SPACE)
50000
40000
30000
20000
10000
0
0
20
40
60
80
100
120
140


Best case performance (SPACE)
50000
40000
30000
20000
10000
0
0
20
40
60
80
100
120
140



**Conclusion:** Honestly I've hated these two algorithms, probably they have an application, but if I have to sort a sequence I will choose Heap sort or Merge sort instead of these two.

The approach is interesting, because is not like comparison based algorithms, we got parameters for sorting and it’s an ingenious way to create a sorting algorithm.