**9주차 과제**

1. **0~999의 세 자리의 숫자로 이루어진 다음 리스트를 기수 정렬을 이용해 정렬하는 과정을 설명하라.**

**[ 10, 123, 56, 636, 992, 119, 234, 76, 82, 345, 567, 432]**

0에서 9까지의 버킷(큐)을 만든다

모든 데이터에 대하여 가장 낮은 자릿수에 해당하는 버킷에 차례대로 데이터를 둔다

0부터 차례대로 버킷에서 데이터를 다시 가져온다

1의 자리부터 시작해서 3의 자리까지 이를 반복한다

Step 1 – [10, 992, 82, 432, 123, 234, 345, 56, 636, 76, 567, 119]

Step 2 – [10, 119, 123, 432, 234, 636, 345, 56, 567, 76, 82, 992]

Step 3 – [10, 56, 76, 82, 119, 123, 234, 345, 432, 567, 636, 992]

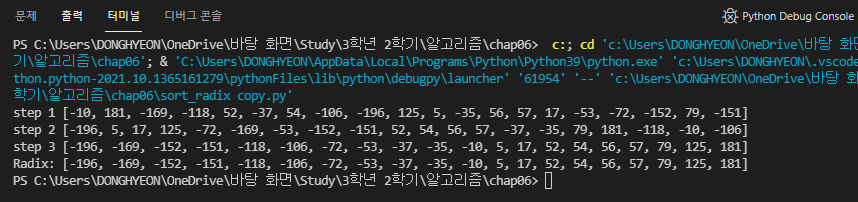
Radix – [10, 56, 76, 82, 119, 123, 234, 345, 432, 567, 636, 992]

**2. –200에서 200사이의 값을 가진 리스트를 기수 정렬하도록 작성하고 소스와 실행 결과를 첨부하세요.**

**# 소스**

|  |
| --- |
| from queue import Queue                 # 파이썬 queue모듈의 Queue 사용  def radix\_sort(A) :      queues = []                     # 큐의 리스트      for i in range(BUCKETS) :          queues.append(Queue())      # BUCKETS개의 큐 사용      n = len(A)      factor = 1                      # 1의 자리부터 시작      for d in range(DIGITS) :            # 모든 자리에 대해          for i in range(n) :         # 자릿수에 따라 큐에 삽입              A[i] += 200              queues[(A[i]//factor) % 10].put(A[i])   # 숫자를 삽입          j = 0          for b in range(BUCKETS) :       # 버킷에서 꺼내어 원래의 리스트로              while not queues[b].empty() : # b번째 큐가 공백이 아닌 동안                  A[j] = queues[b].get()  # 원소를 꺼내 리스트에 저장                  j += 1          for i in range(n):              A[i] -= 200          factor \*= 10                    # 그 다음 자리수로 간다.          print("step", d+1, A)           # 중간 과정 출력용 문장    import random       # 테스트를 위한 난수 발생을 위해 random 모듈 포함  BUCKETS = 10        # 10진법으로 정렬  DIGITS  = 3     # 최대 3 자릿수  data = []  for i in range(20) :      data.append(random.randint(-200, 200))  # -200~200사이의 숫자 20개 생성  radix\_sort(data)                        # 기수 정렬  print("Radix:", data)                   # 결과 출력 |

**# 실행 결과**

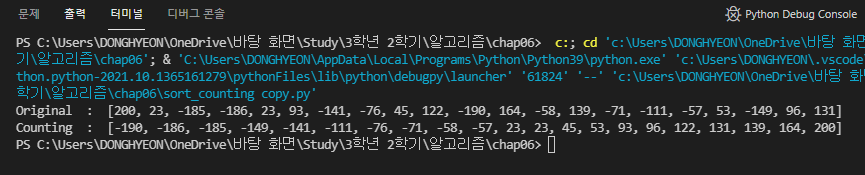


**3. –200에서 200사이의 값을 가진 리스트를 카운팅 정렬하도록 작성하고 소스와 실행 결과를 첨부하세요.**

**# 소스**

|  |
| --- |
| import random  def counting\_sort(A):      output = [0] \* len(A)               # 정렬 결과저장용 임시 리스트      count  = [0] \* MAX\_VAL              # 각 숫자의 빈도를 저장      for i in range(len(A)):          A[i] += 200      for i in A:                         # 각 숫자별 빈도를 계산          count[i] += 1        for i in range(MAX\_VAL):            # count[i]가 출력 배열에서          count[i] += count[i-1]          # 해당 숫자의 위치가 되도록 수정        for i in range(len(A)):             # 모든 입력항목 A[i]에 대해          output[count[A[i]]-1] = A[i]    # 해당 위치(count[A[i]]-1)에 저장          count[A[i]] -= 1                # 킷값 A[i]의 위치를 하나 줄임      for i in range(len(A)):             # 정렬 결과를 원래 배열에 복사          output[i] -= 200          A[i] = output[i]  MAX\_VAL = 402  data = []  data.append(200)  for i in range(20) :      data.append(random.randint(-200, 200))  # -200~200사이의 숫자 20개 생성  print("Original  : ", data)  counting\_sort(data)             # 카운팅 정렬  print("Counting  : ", data) |

**# 실행 결과**



**4. 문자열 매칭 알고리즘은 DNA 염기서열 탐색에 사용될 수 있다. DNA는 {A,T,G,C} 만으로 구성되어 있다.**

**1) 다음과 같은 염기서열에 대한 시프트 테이블을 구성하라**

**TCCTATTCTT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 문자 | A | C | G | T |
| shift | 5 | 2 | 10 | 1 |

**(2) 호스풀 알고리즘을 이용해 다음 DNA 시퀀스에서 위의 염기서열을 찾아라.**

**TTATAGATCTCGTATTCTTTTATAGATCTCCTATTCTT**

**# 실행 결과**

**TTATAGATCTCGTATTCTTTTATAGATCTCCTATTCTT**

**TCCTATTCTT (T 일치 C 불일치 2칸 – 1칸 시프트)**

**TCCTATTCTT (C 불일치 2칸 시프트)**

**TCCTATTCTT (T 일치 G 불일치 10칸 – 1칸 시프트)**

**TCCTATTCTT (A 불일치 5칸 시프트)**

**TCCTATTCTT (T 일치 A 불일치 5칸 – 1칸 시프트)**

**TCCTATTCTT (C 불일치 2칸 시프트)**

**TCCTATTCTT (A 불일치 5칸 시프트)**

**TCCTATTCTT (TCCTATTCTT까지 일치)**

**1 + 2 + 9 + 5 + 4 + 2 + 5**

**패턴의 위치 : 28**

**5. 크기가 11인 해시 테이블을 가정하자. 해시 함수로는 다음을 사용한다.**

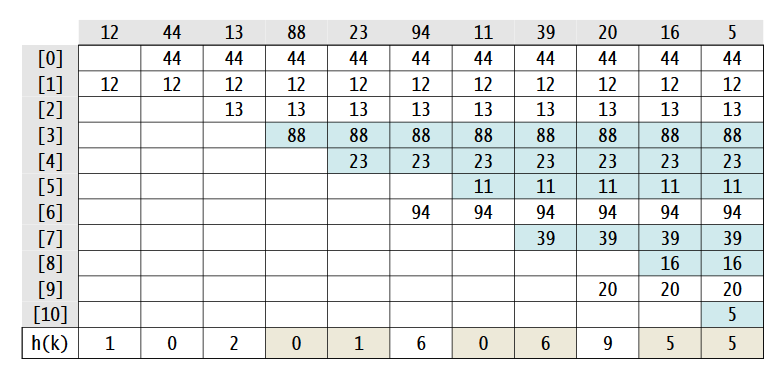
**h(k) = k mod 11**

**입력 자료가 다음과 같은 순서로 입력된다고 하면 다음 각 경우에 대한 해시 테이블의 내용을 그려라.**

**12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5**

**(1) 충돌을 선형 조사법을 사용하여 처리한다.**

**(정답제공 나머지도 아래와 같이 작성하세요.)**

****

**(2) 충돌을 이차 조사법을 사용하여 처리한다.**

**충돌시 이차조사법 : h’(k)= (k+j\*j)%11**

12 44 13 88 23 94 11 39 20 16 5

[0] 39 39 39 39

[1] 12 12 12 12 12 12 12 12 12 12 12

[2] 44 44 44 44 44 44 44 44 44 44

[3] 11 11 11 11 11

[4] 16 16

[5] 5

[6] 13 13 13 13 13 13 13 13 13

[7] 23 23 23 23 23 23 23

[8] 20 20 20

[9] 88 88 88 88 88 88 88 88

[10] 94 94 94 94 94 94

h’(k) 1 1 6 9 6 9 3 0 7 9 6

**(3) 충돌을 다음과 같은 이중 해시법을 사용하여 처리한다.**

**h'(k) = 7—(k mod 7)**

12 44 13 88 23 94 11 39 20 16 5

[0] 16 16

[1] 5

[2] 12 12 12 12 12 12 12 12 12 12 12

[3] 23 23 23 23 23 23 23

[4] 88 88 88 88 88 88 88 88

[5] 44 44 44 44 44 44 44 44 44 44

[6] 13 13 13 13 13 13 13 13 13

[7] 94 94 94 94 94 94

[8] 11 11 11 11 11

[9] 39 39 39 39

[10] 20 20 20

h’(k) 2 5 6 4 2 3 4 4 6 2 5

**⑷ 충돌을 체인법을 사용하여 처리한다.**

12 44 13 88 23 94 11 39 20 16 5

[0] 44 44 44 44 44 44 44 44 44 44

88 88 88 88 88 88 88 88

11 11 11 11 11

[1] 12 12 12 12 12 12 12 12 12 12 12

23 23 23 23 23 23 23

[2] 13 13 13 13 13 13 13 13 13

[3]

[4]

[5] 16 16

5

[6] 94 94 94 94 94 94

39 39 39 39

[7]

[8]

[9] 20 20 20

[10]

h(k) 1 0 2 0 1 6 0 6 9 5 5