동국대학교

Lab 4 알고리즘 레포트

|  |  |
| --- | --- |
| 이름 | 한윤수 |
| 학과 | 컴퓨터AI학부 |
| 학번 | 2021113405 |
| 제출일자 | 2025.10.01 |

1. 임의의 자연수 N을 입력 받아서 N개의 랜덤한 데이터를 생성해 반복문과 조건문 만을 이용하여 힙(heap)정렬의 각 단계별 결과를 보여주는 코드를 작성하고 반드시 코드에 대한 자세한 설명을 추가하시오.

입력 예제

생성할 랜덤 변수 개수를 입력하세요: 10

출력 예제

생성된 list : [63, 86, 47, 89, 76, 42, 69, 35, 11, 36]

heap root : 86

정렬 단계: [86, 76, 69, 63, 36, 42, 47, 35, 11, 89]

heap root : 76

정렬 단계: [76, 63, 69, 35, 36, 42, 47, 11, 86, 89]

heap root : 69

정렬 단계: [69, 63, 47, 35, 36, 42, 11, 76, 86, 89]

heap root : 63

정렬 단계: [63, 36, 47, 35, 11, 42, 69, 76, 86, 89]

heap root : 47

정렬 단계: [47, 36, 42, 35, 11, 63, 69, 76, 86, 89]

heap root : 42

정렬 단계: [42, 36, 11, 35, 47, 63, 69, 76, 86, 89]

heap root : 36

정렬 단계: [36, 35, 11, 42, 47, 63, 69, 76, 86, 89]

heap root : 35

정렬 단계: [35, 11, 36, 42, 47, 63, 69, 76, 86, 89]

heap root : 11

정렬 단계: [11, 35, 36, 42, 47, 63, 69, 76, 86, 89]

최종 정렬 결과: [11, 35, 36, 42, 47, 63, 69, 76, 86, 89]

1.1 소스코드 및 설명

|  |
| --- |
| import random  def main():      test = True      if test:          array\_size = 10          array = [63, 86, 47, 89, 76, 42, 69, 35, 11, 36]      else:          array\_size = int(input("생성할 랜덤 변수 개수를 입력하세요: "))          array = random.sample(range(1, 101), array\_size)      ans = []      ans.append("생성된 list : [" + ", ".join(map(str, array)) + "]")      sorted\_array = heap\_sort(array, ans)      ans.append("최종 정렬 결과: [" + ", ".join(map(str, sorted\_array)) + "]")      final\_ans = "\n".join(ans)      print(final\_ans)  def heap\_sort(array, ans):      n = len(array)      # 1단계: 최대 힙(Max Heap) 만들기      for i in range(n // 2 - 1, -1, -1):          heapify(array, n, i)      # 2단계: 힙에서 하나씩 꺼내 정렬      for end in range(n - 1, 0, -1):          # root(최대값)와 마지막 원소 교환          array[0], array[end] = array[end], array[0]          # 교환 후 힙 성질 복구          heapify(array, end, 0)          # heap root 저장          ans.append(f"heap root : {array[0]}")          # 현재 단계 결과 저장          ans.append("정렬 단계: [" + ", ".join(map(str, array)) + "]")      return array  def heapify(array, n, i):      largest = i  # 현재 노드를 최대값으로 가정      left = 2 \* i + 1  # 왼쪽 자식 인덱스      right = 2 \* i + 2  # 오른쪽 자식 인덱스      # 왼쪽 자식이 존재하고 현재 노드보다 크다면      if left < n and array[left] > array[largest]:          largest = left      # 오른쪽 자식이 존재하고 현재 노드보다 크다면      if right < n and array[right] > array[largest]:          largest = right      # 최대값이 현재 노드가 아니라면 교환 후 재귀적으로 heapify      if largest != i:          array[i], array[largest] = array[largest], array[i]          heapify(array, n, largest)  if \_\_name\_\_ == "\_\_main\_\_":      main() |

1.2 설명

임의의 자연수 N개의 데이터를 생성한 뒤 힙 정렬을 수행하며 각 단계의 결과를 출력한다. 먼저 배열을 최대 힙 구조로 변환한 후, 루트(가장 큰 값)를 배열 끝으로 보내고 힙의 크기를 줄여가면서 다시 힙 속성을 유지하는 과정을 반복한다. 각 단계에서는 현재 힙의 루트 값과 교환 및 재정렬된 배열 상태를 출력하여 힙 정렬이 점진적으로 정렬된 결과로 수렴하는 과정을 확인할 수 있다.

1.3 결과 화면

|  |
| --- |
|  |

2. 임의의 자연수 N을 입력 받아서 N개의 랜덤한 데이터를 생성해 반복문과 조건문 만을 이용하여 계수정렬의 각 단계별 결과를 보여주는 코드를 작성하고 반드시 코드에 대한 자세한 설명을 추가하시오. 단, 이때 데이터의 범위는 단계별 원소 개수의 배열과 단계별 배열을 확인하기 위해 1~10으로 제한한다. N의 개수는 최소 10이상을 입력

입력 예제

생성할 랜덤 변수 개수를 입력하세요: 10

출력 예제

생성된 list : [4, 8, 4, 4, 4, 2, 7, 1, 3, 5]

초기 원소의 개수 배열 : [1, 1, 1, 4, 1, 0, 1, 1]

원소의 개수 배열 : [0, 1, 1, 4, 1, 0, 1, 1]

단계별 배열 : [1, 8, 4, 4, 4, 2, 7, 1, 3, 5]

원소의 개수 배열 : [0, 0, 1, 4, 1, 0, 1, 1]

단계별 배열 : [1, 2, 4, 4, 4, 2, 7, 1, 3, 5]

원소의 개수 배열 : [0, 0, 0, 4, 1, 0, 1, 1]

단계별 배열 : [1, 2, 3, 4, 4, 2, 7, 1, 3, 5]

원소의 개수 배열 : [0, 0, 0, 0, 1, 0, 1, 1]

단계별 배열 : [1, 2, 3, 4, 4, 4, 4, 1, 3, 5]

원소의 개수 배열 : [0, 0, 0, 0, 0, 0, 1, 1]

단계별 배열 : [1, 2, 3, 4, 4, 4, 4, 5, 3, 5]

원소의 개수 배열 : [0, 0, 0, 0, 0, 0, 1, 1]

단계별 배열 : [1, 2, 3, 4, 4, 4, 4, 5, 3, 5]

원소의 개수 배열 : [0, 0, 0, 0, 0, 0, 0, 1]

단계별 배열 : [1, 2, 3, 4, 4, 4, 4, 5, 7, 5]

원소의 개수 배열 : [0, 0, 0, 0, 0, 0, 0, 0]

단계별 배열 : [1, 2, 3, 4, 4, 4, 4, 5, 7, 8]

최종 정렬 결과: [1, 2, 3, 4, 4, 4, 4, 5, 7, 8]

2.1 소스코드 및 설명

|  |
| --- |
| import random  def main():      test = True      if test:          array\_size = 10          array = [4, 8, 4, 4, 4, 2, 7, 1, 3, 5]      else:          array\_size = int(input("생성할 랜덤 변수 개수를 입력하세요 (최소 10): "))          array = [random.randint(1, 10) for \_ in range(array\_size)]      ans = []      ans.append("생성된 list : [" + ", ".join(map(str, array)) + "]")      sorted\_arr = counting\_sort\_step(array, ans)      ans.append("최종 정렬 결과: [" + ", ".join(map(str, sorted\_arr)) + "]")      final\_ans = "\n".join(ans)      print(final\_ans)  def counting\_sort\_step(arr, ans):      max\_val = 10  # 값의 범위 (1~10)      # 1단계: count 배열 초기화      count = [0] \* max\_val      for num in arr:          count[num - 1] += 1  # 1~10 → 인덱스 0~9      ans.append("초기 원소의 개수 배열 : [" + ", ".join(map(str, count)) + "]")      sorted\_arr = arr[:]  # 원본 배열 복사해서 단계별 교체      # 2단계: count 배열 순회하면서 정렬      index = 0      for i in range(max\_val):          while count[i] > 0:              count[i] -= 1              sorted\_arr[index] = i + 1  # 실제 값은 인덱스+1              index += 1              # 단계별 출력              ans.append("원소의 개수 배열 : [" + ", ".join(map(str, count)) + "]")              ans.append("단계별 배열 : [" + ", ".join(map(str, sorted\_arr)) + "]")      return sorted\_arr  if \_\_name\_\_ == "\_\_main\_\_":      main() |

2.2 설명

1~10 범위의 난수를 N개 생성한 뒤 계수 정렬을 수행하며 각 단계의 결과를 보여준다. 먼저 count 배열을 만들어 각 숫자가 등장하는 횟수를 기록하고, 이를 바탕으로 원래 배열을 단계적으로 채워나간다. 정렬 과정에서 count 배열의 감소와 배열에 값이 채워지는 모습을 매 단계마다 출력하여 계수 정렬이 데이터의 빈도 기반으로 정렬을 완성하는 과정을 직관적으로 확인할 수 있다.

2.3 결과 화면

|  |
| --- |
|  |

3. 임의의 자연수 N을 입력 받아서 N개의 랜덤한 데이터를 생성해 반복문과 조건문 만을 이용하여 기수정렬의 각 단계별 결과를 보여주는 코드를 작성하고 반드시 코드에 대한 자세한 설명을 추가하시오. 단, 이때 데이터의 범위는 단계별 원소 개수의 배열과 단계별 배열을 확인하기 위해 1~10000으로 제한 한다. N의 개수는 최소 20이상을 입력

입력 예제

생성할 랜덤 변수 개수를 입력하세요: 20

출력 예제

생성된 list : [4785, 8642, 7638, 5502, 7276, 3657, 7246, 2591, 3770, 4156, 6807, 730, 9999, 6405, 7594, 1003, 605, 8390, 9636, 3716]

자릿수 1 단계: [3770, 730, 8390, 2591, 8642, 5502, 1003, 7594, 4785, 6405, 605, 7276, 7246, 4156, 9636, 3716, 3657, 6807, 7638, 9999]

자릿수 10 단계: [5502, 1003, 6405, 605, 6807, 3716, 730, 9636, 7638, 8642, 7246, 4156, 3657, 3770, 7276, 4785, 8390, 2591, 7594, 9999]

자릿수 100 단계: [1003, 4156, 7246, 7276, 8390, 6405, 5502, 2591, 7594, 605, 9636, 7638, 8642, 3657, 3716, 730, 3770, 4785, 6807, 9999]

자릿수 1000 단계: [605, 730, 1003, 2591, 3657, 3716, 3770, 4156, 4785, 5502, 6405, 6807, 7246, 7276, 7594, 7638, 8390, 8642, 9636, 9999]

최종 정렬 결과: [605, 730, 1003, 2591, 3657, 3716, 3770, 4156, 4785, 5502, 6405, 6807, 7246, 7276, 7594, 7638, 8390, 8642, 9636, 9999]

3.1 소스코드 및 설명

|  |
| --- |
| import random  def main():      test = True      if test:          array\_size = 20          array = [4785, 8642, 7638, 5502, 7276, 3657, 7246, 2591, 3770, 4156, 6807, 730, 9999, 6405, 7594, 1003, 605, 8390, 9636, 3716]      else:          array\_size = int(input("생성할 랜덤 변수 개수를 입력하세요 (최소 20): "))          array = [random.randint(1, 10000) for \_ in range(array\_size)]      ans = []      ans.append("생성된 list : [" + ", ".join(map(str, array)) + "]")      sorted\_arr = radix\_sort(array, ans)      ans.append("최종 정렬 결과: [" + ", ".join(map(str, sorted\_arr)) + "]")      final\_ans = "\n".join(ans)      print(final\_ans)  def radix\_sort(arr, ans):      # 최대값 구해서 자릿수 확인      max\_num = max(arr)      exp = 1  # 1의 자리부터 시작      while max\_num // exp > 0:          counting\_sort\_by\_digit(arr, exp, ans)          exp \*= 10      return arr  def counting\_sort\_by\_digit(arr, exp, ans):      n = len(arr)      output = [0] \* n      count = [0] \* 10  # 자릿수는 0~9      # 1단계: 현재 자리수의 count 배열 만들기      for i in range(n):          digit = (arr[i] // exp) % 10          count[digit] += 1      # 2단계: 누적합 (자리별 위치 결정)      for i in range(1, 10):          count[i] += count[i - 1]      # 3단계: output 배열 채우기 (안정 정렬 유지)      i = n - 1      while i >= 0:          digit = (arr[i] // exp) % 10          output[count[digit] - 1] = arr[i]          count[digit] -= 1          i -= 1      # 4단계: 결과를 원본 arr에 반영      for i in range(n):          arr[i] = output[i]      # 현재 자릿수 단계 결과 저장      ans.append(f"자릿수 {exp} 단계: [" + ", ".join(map(str, arr)) + "]")  if \_\_name\_\_ == "\_\_main\_\_":      main() |

3.2 설명

1~10000 범위의 난수를 최소 20개 이상 생성하고, 기수 정렬을 수행하며 각 자릿수 단계별 정렬 결과를 출력한다. 정렬은 가장 낮은 자릿수(1의 자리)부터 시작하여 점차 높은 자릿수(10, 100, 1000 자리)로 확장되며, 각 단계에서는 안정성을 유지하기 위해 자릿수 기준으로 계수 정렬을 적용한다. 이를 통해 각 자릿수별 재배치 과정을 출력하고, 최종적으로 전체 배열이 오름차순으로 완전히 정렬되는 과정을 확인할 수 있다.

3.3 결과 화면

|  |
| --- |
|  |

4. 임의의 자연수 N을 입력 받아서 N개의 랜덤한 데이터를 생성해 반복문과 조건문 만을 이용하여 버킷정렬의 각 단계별 결과를 보여주는 코드를 작성하고 반드시 코드에 대한 자세한 설명을 추가하시오.

입력 예제

생성할 랜덤 변수 개수를 입력하세요: 10

출력 예제

생성된 list : [78, 49, 91, 91, 46, 57, 52, 99, 73, 73]

단계별 버킷들 : [[], [], [], [], [], [], [78], [], [], []]

단계별 버킷들 : [[49], [], [], [], [], [], [78], [], [], []]

단계별 버킷들 : [[49], [], [], [], [], [], [78], [], [91], []]

단계별 버킷들 : [[49], [], [], [], [], [], [78], [], [91, 91], []]

단계별 버킷들 : [[49, 46], [], [], [], [], [], [78], [], [91, 91], []]

단계별 버킷들 : [[49, 46], [], [57], [], [], [], [78], [], [91, 91], []]

단계별 버킷들 : [[49, 46], [52], [57], [], [], [], [78], [], [91, 91], []]

단계별 버킷들 : [[49, 46], [52], [57], [], [], [], [78], [], [91, 91], [99]]

단계별 버킷들 : [[49, 46], [52], [57], [], [], [73], [78], [], [91, 91], [99]]

단계별 버킷들 : [[49, 46], [52], [57], [], [], [73, 73], [78], [], [91, 91], [99]]

최종 정렬 결과: [46, 49, 52, 57, 73, 73, 78, 91, 91, 99]

4.1 소스코드 및 설명

|  |
| --- |
| import random  def bucket\_sort(arr, ans):      bucket\_count = 10  # 버킷 개수 고정      min\_val, max\_val = min(arr), max(arr)      # 버킷 크기를 실수로 계산      bucket\_size = (max\_val - min\_val) / bucket\_count      # 빈 버킷 생성      buckets = [[] for \_ in range(bucket\_count)]      # 1단계: 각 데이터를 버킷에 분배      for num in arr:          # 버킷 인덱스 = (값 - min) / 버킷크기          idx = int((num - min\_val) / bucket\_size)          if idx >= bucket\_count:  # max 값은 마지막 버킷에 강제로 배치              idx = bucket\_count - 1          buckets[idx].append(num)          ans.append("단계별 버킷들 : " + str(buckets))      # 2단계: 각 버킷 내부 정렬 (삽입 정렬)      for b in buckets:          for i in range(1, len(b)):              key = b[i]              j = i - 1              while j >= 0 and b[j] > key:                  b[j + 1] = b[j]                  j -= 1              b[j + 1] = key      # 3단계: 모든 버킷 합치기      sorted\_arr = []      for b in buckets:          sorted\_arr.extend(b)      return sorted\_arr  def main():      test = True      if test:          array = [78, 49, 91, 91, 46, 57, 52, 99, 73, 73]      else:          array\_size = int(input("생성할 랜덤 변수 개수를 입력하세요 (최소 10): "))          array = [random.randint(1, 100) for \_ in range(array\_size)]      ans = []      ans.append("생성된 list : " + str(array))      sorted\_arr = bucket\_sort(array, ans)      ans.append("최종 정렬 결과: " + str(sorted\_arr))      print("\n".join(ans))  if \_\_name\_\_ == "\_\_main\_\_":      main() |

4.2 설명

난수 배열을 입력받아 버킷 정렬을 수행하며 각 단계별 버킷 상태를 출력한다. 전체 데이터의 최솟값과 최댓값을 이용해 버킷 크기를 실수로 계산한 뒤, 각 원소를 해당 범위에 맞는 버킷에 분배한다. 각 원소가 추가될 때마다 현재 버킷들의 상태를 출력하며, 이후 각 버킷을 삽입 정렬로 정렬하고 모든 버킷을 합쳐 최종 정렬된 결과를 만든다. 이 과정을 통해 버킷 정렬이 구간 기반 분배와 부분 정렬을 통해 전체 정렬을 달성하는 원리를 단계적으로 확인할 수 있다.

4.3결과 화면

|  |
| --- |
|  |