**Euclidean algorithm**

*Euclidean algorithm을 이용한 최소 공략 법을 구하는 방법*

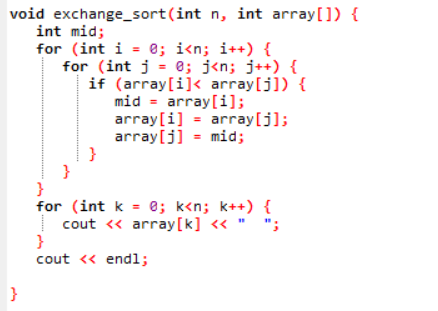
1. 넘어온 인자 a와 b 중 큰 수와 작은 수를 비교하여 큰 수에서 작은 수를 빼 다시 큰 수에 넣는다
2. a 와 b가 같아 질때까지 1을 반복
3. a 와 b가 같아 졌다면 두 숫자 중 하나를 리턴

*둘을 서로 나눈 후 나머지 값을 반환하여 서로 나뉘지 않을 때까지 반복하는 최소 공략법을 구하는 방법*

1. 넘어온 인자 a와 b를 나눈 나머지 값을 임시 배열에 저장
2. a 와 b를 교환
3. b에 나머지 값을 저장
4. b 가 0이 될 때 까지 반복

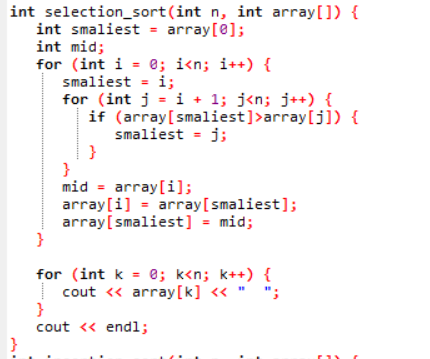
*위의 방법을 재귀 함수를 통해 표현*

**정렬**

**Exchange sort**

*인덱스 키와 나머지 키를 비교하여 인덱스 키의 크기가 나머지 키보다 크면 서로 순서를 바꾸는 정렬 방법*

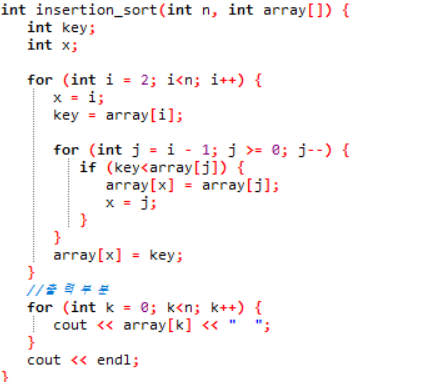
1. 인덱스 키를 표시할 바깥쪽 반복문과 인덱스 키와 비교할 나머지 키를 반복할 안쪽 반복문을 선언 한다.
2. 반복문을 도는 도중 인덱스 키가 현재 차례인 키보다 크면 서로 자리를 바꾼다.
3. 배열에 저장된 값을 출력한다.



**selection sort**

*주어진 배열 중에서 최소 값을 찾아 앞에 위치시키는 정렬 방법*

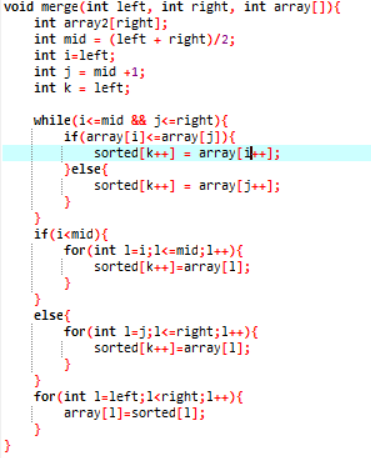
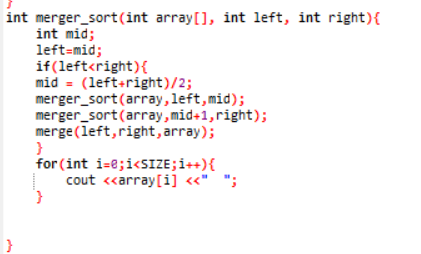
1. 주어진 배열 값에서 최소 값을 찾는다
2. 맨 처음 값과 교환한다
3. 맨 처음 값을 뺀 나머지 배열을 비교하여 최소값을 찾아 두번째 자리에 넣는다
4. 인덱스가 맨 마지막 순서까지 도달하면 반복문이 종료 된다.

**insertion sort**

*인덱스 키를 정한 후 해당 인덱스 키가 현재 선택 된 키보다 크지만 앞에 있을 경우 선택 된 키를 인덱스 앞에 삽입하는 정렬*

1. 배열의 첫번째를 나타내는 x를 선언한다.
2. 전체 배열을 순회하며 key값보다 클 경우 현재 key 값을 가지고 있는 배열 array[x]를 옆으로 미뤄준다. key값보다 작을 경우 x는 변하지 않는다
3. 내부 반복문이 끝난 후 빈자리 x에 해당 key 값을 넣는다.

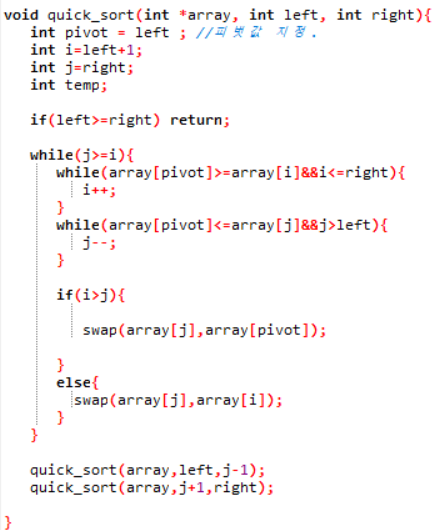
**merger sort**



*분할 정복 알고리즘을 이용한 정렬*

1. 배열의 길이가 0또는 1이면 이미 정렬된 것으로 본다 그렇지 않은 경우 배열을 절반으로 자른다.
2. 재귀를 통해 배열의 길이가 0 또는 1 분할 후 이 배열을 정렬한다.
3. 분할 되어 정렬된 배열 두개를 다시 정렬한다.
4. 이후 나눠진 배열을 합병한다.
5. 모든 배열이 합병 될 때까지 위 과정을 반복한다.

메소도는 분할하는 메소드와 합병하는 메소드 두개로 이루어져 있다.

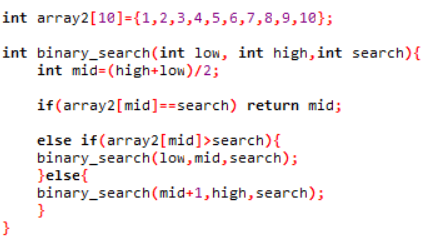
**quick sort**

*분할 정복 알고리즘을 이용한 정렬 방법 중 하나로 매우 빠른 속도를 자랑하며 합병 정렬과는 달리 배열을 비균등하게 분할한다.*

1. 배열안에 있는 한 요소를 pivot으로 선택한다
2. Pivot 기준으로 피벗보다 작은 요소들은 모두 피벗 왼쪽으로 옮겨지고 피벗보다 큰 요소들은 모두 피벗의 오른쪽으로 옮겨진다
3. 피벗을 제외한 왼쪽 리스트와 오른쪽 리스트를 다시 정렬한다.
4. 위 과정을 배열이 더 이상 분할이 불가능 할 때까지 반복한다.

**Binary search**

*정렬 되어 있는 배열에서 특정 값을 찾을 때 사용하는 알고리즘으로 중간 값을 설정 한 뒤 찾으려는 값이 중간 값 보다 작으면 왼쪽 크면 오른 쪽 배열의 중간 값을 찾아가 값을 찾을 때 까지 반복하는 서치 알고리즘이다*

1. 배열의 중간 값과 찾으려는 키 값을 비교한 후 같으면 그 수를 리턴 한다
2. 만약 배열 중간 값보다 키 값이 크다면 재귀 함수를 이용하여 오른 쪽 중간 값과 키 값이 일치하는지 비교하 고 작다면 왼쪽 배열 값의 중간 값과 비교한다.
3. 원하는 값을 찾을 때 까지 재귀를 반복 한다.

**DP와 Greedy**

*Greedy :*

*과거에 일어났던 일이나 미래의 일어날 선택을 고려하지 않고 현재 상태에서 최상의 값을 고르는 방법*

*DP :*

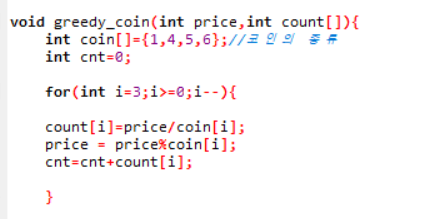
*Memoization을 이용하여 과거의 일어났던 선택을 참고하는 것*

*When can you use it?*

*아래 조건을 만족 해야 사용 가능*

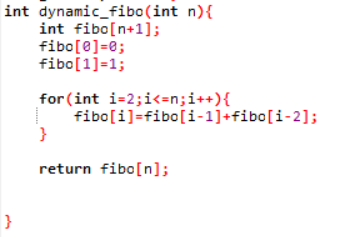
*optimal substructure : 지역단위의 작은 문제로부터 최상의 솔루션을 조합하여 최적의 값을 구할 수 있어야한다.*

*overlapping subproblem :최적의 값을 찾는 방법은 같은 문제를 푼 경우가 여러 번 포함 되어야 한다.)*

**Greedy : 동전교환 문제**

*Greedy를 이용하여 가장 동전의 개수가 적게 거스름 돈을 거슬러 받는 문제를 푼 것*

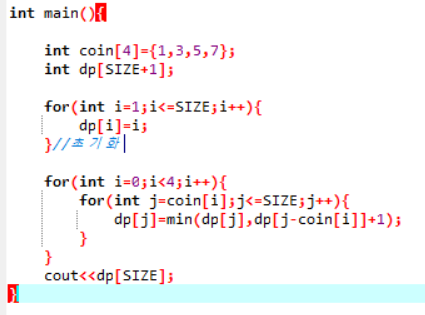
1. 반복문을 사용하여 각각 코인이 큰 순서대로 나눠 몫은 count 배열에 나머지는 남은 price에 넣는다
2. 위 과정을 코인의 종류별로 한번씩 반복한다.

**DP를 사용한 피보나치 수열**

*DP를 이용하한 피보나치 수열*

1. fibo라는 n 크기의 배열을 선언한다
2. fibo[0]을 0 , fibo[1]를 1로 초기화 시켜 준다.
3. fibo 배열의 메모라이징 된 수를 이용하여 다음 fibo 수에 i-1 i-2번쨰 fibo수를 더한 수를 메모라이징 시켜준다.
4. 구하고 싶은 n번째 피보나치 수 만큼 반복시켜준다.

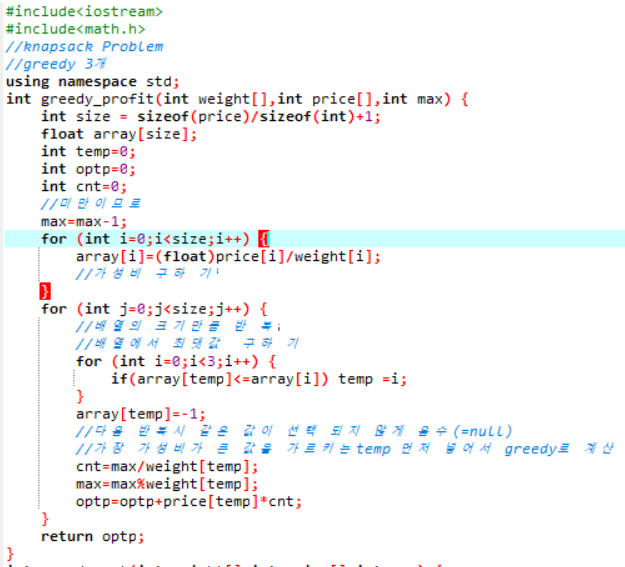
**DP : 동전교환 문제**

*DP를 이용하여 가장 동전의 개수가 적게 거스름 돈을 거슬러 받는 문제를 푼 것*

점화식 dp[j] = min( dp[j] , dp[j-coin[i]] ) 을 이용하여 푸는 방법으로

Dp배열에는 각 코인별로 돌려받을 수 있는 거스름돈의 개수가 들어간다.

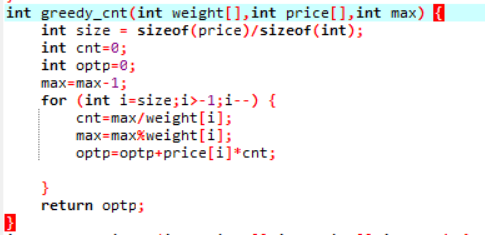
따라서, 여기서 dp[j]는 이전의 동전 (i-1)을 기준으로 누적시킨 경우의 수이고 dp[j-coin[i]]은 현재 동전 I 를 사용하여 만들 수 있는 경우의 수를 나타난데 이 둘을 비교하여 작은 값을 dp에 memorization하고 다음 연산 때 이 dp가 연산에 포함되어 최솟값을 찾는다.

**가중치 중심 greedy**

*Greedy를 이용하여 가성비가 큰 순서대로 선택해 가방문제를 푼 것*

1. 임시 배열에 price/weight를 하여 가성비를 구하여 준다.
2. 최대 들 수 있는 무게에서 가장 가성비가 큰 순서대로 차례로 가성비에 해당하는 무게로 나누어 준 후 몫은 가격 과 곱한 후 지금 까지 훔친 가치변수에 저장한다..
3. 나눈 나머지는 들 수 있는 남은 무게에 다시 삽입하여 준다.
4. 위 과정을 가격 배열의 크기만큼 반복한다

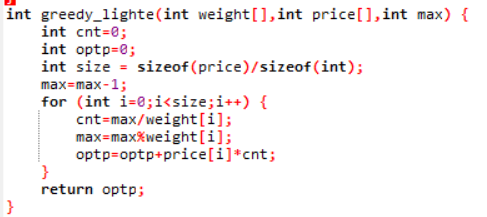
**가격 중심 greedy**

*Greedy를 이용하여 가격이 비싼 순서대로 선택해 가방문제를 푼 것*

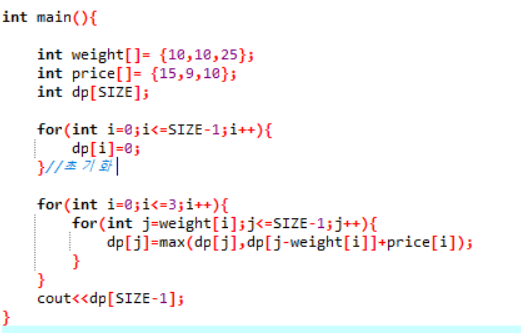
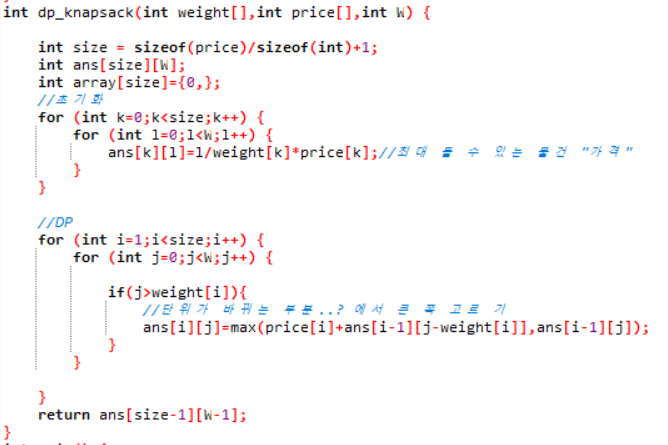
1. 가격 배열을 내림 차순으로 정렬해 준다.
2. 해당 가격에 해당되는 물건의 무게로 들 수 있는 무게를 나누어 나눈 몫은 해당 가격과 곱하고 나눈 나머지는 들 수 있는 남은 무게로써 다시 삽입하여 준다.
3. 위 과정을 가격 배열의 크기만큼 반복한다.

**가벼운 순서 greedy**

*Greedy를 이용하여 무게가 가벼운 순서대로 선택해 가방문제를 푼 것*

1. 무게 배열을 내림 차순으로 정렬해 준다.
2. 무게가 가벼운 순서대로 들 수 있는 무게를 나누어 나눈 후 몫은 해당 가격과 곱하고 나눈 나머지는 들 수 있는 남은 무게로써 다시 삽입하여 준다.
3. 위 과정을 가격 배열의 크기만큼 반복한다.

**DP**



*DP를 사용하여 가방 문제를 푼 것*

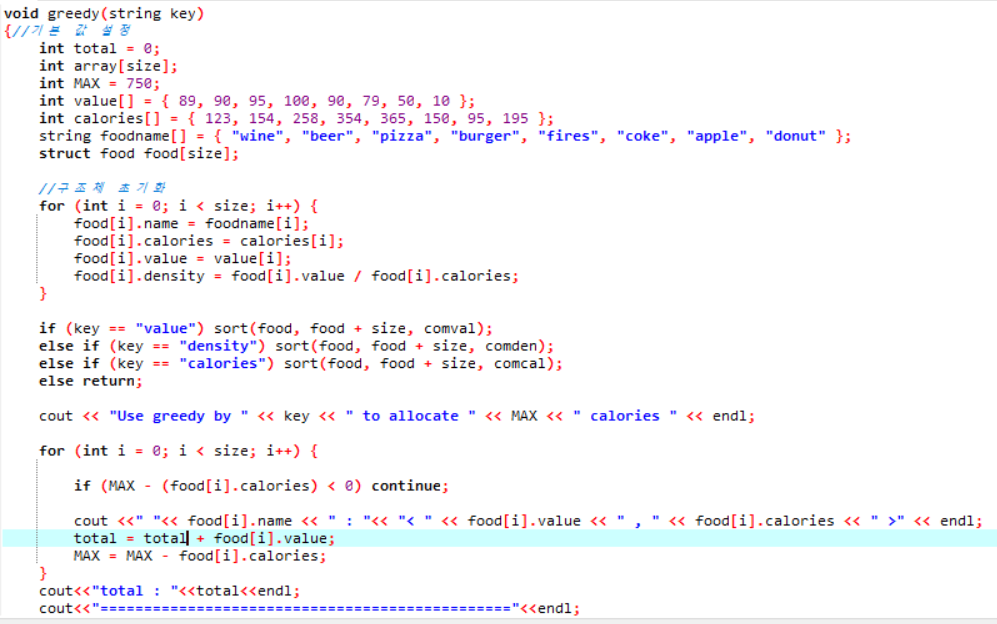
(위)점화식 : max(price[i]+ams[i-1][j-weight[i]], ans[i-1][j] ) 을 이용하여 가방 문제를 푼다.

(아래) 점화식 : max(dp[j],dp[j-weight[i]]+price[i]) 을 이용하여 가방 문제를 푼다.

DP을 이용한 동전 문제 와 같이 새로 구한 부분인 price[i]+ams[i-1][j-weight[i]]와 예전에 memorization 해 두었던 값 ans[i-1][j]를 비교하여 더 작은 값을 ans[i][j]에 새로이 memorization 하는 방법이다.

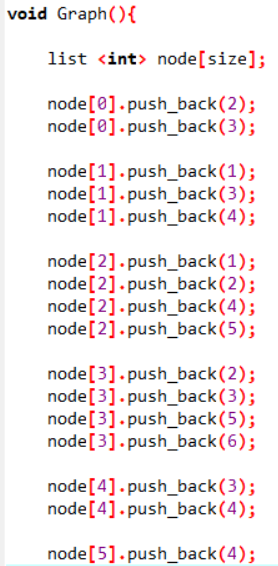
(아래의 점화식도 같은 memorization을 이용하여 푼다.)

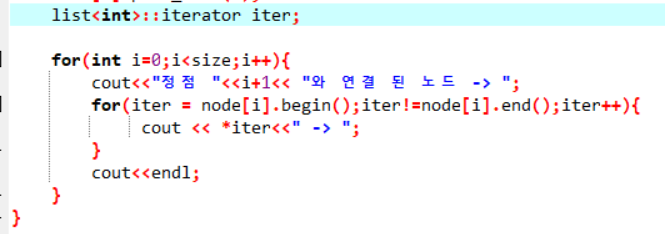
**Greedy 칼로리 계산(재사용성을 높임)**



*위의 greedy 문제들을 수정하기 쉽게 객체지향적으로 바꾼 코드*

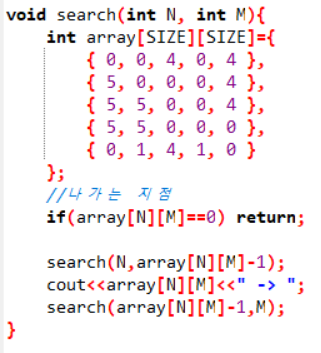
**Graph**

**그래프 만들기**



(리스트 출력문)

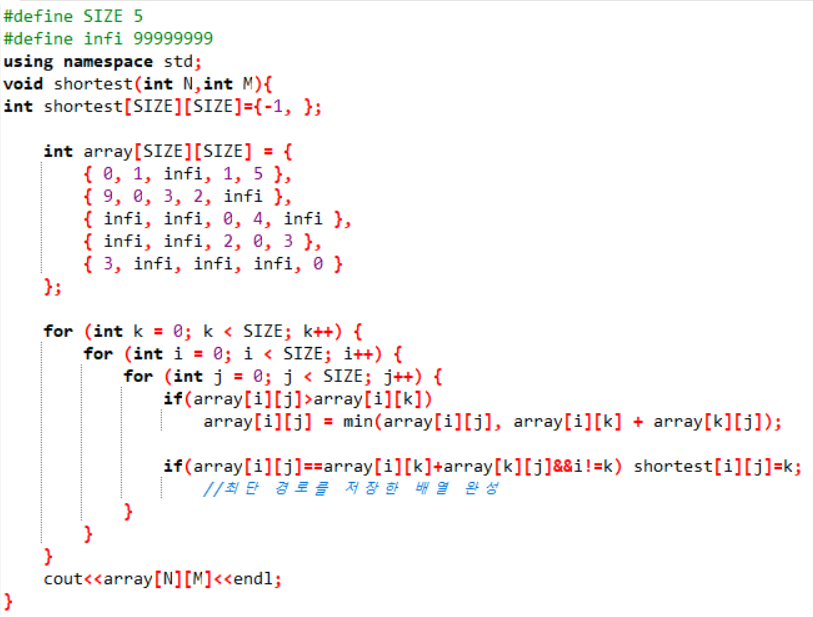
1. 리스트를 배열로 선언한다
2. 리스트 하나는 하나의 노드를 의미하며 push\_back이라는 list 내장 함수를 통해 해당 리스트가 어떤 노드와 연결 되어 있는지 값을 넣어 준다.
3. 출력

**최단 경로 구하기**

*재귀를 이용한 최단 경로를 구하는 알고리즘*

1. 행 노드에서 열 노드로 가는 최단 노드를 저장한다. (이때 거쳐서 가는 것이 아닌 직접 행 노드와 열 노드에서 직접 연결되어 가는 게 최단 거리 일 때는 해당 배열 값에 0을 삽입한다)
2. 재귀를 통해 만약 해당 최단 노드가 0일 경우에는 함수를 종료하지만 아닐 경우 해당 처음 -> 거쳐가는 노드와 거쳐가는 노드 -> 도착 노드 최단 값을 구하기 위하여 재귀를 이용한다.

**최단 거리 구하기**



*3중 반복문을 이용한 최단 거리를 구하는 알고리즘*

1. 2차 배열을 선언하여 행 노드에서 열 노드로 가는 가중치 값을 저장한다(여기서 inif는 아주 큰 값을 의미한다.).
2. 2차 배열을 순회하며 비교하기 위한 반복문 두개와 거쳐가는 노드를 지정해 주기위한 반복문 하나를 통해 k로 거쳐가는 거리가 더 짧은 지 거쳐 가지 않고 원래 있던 경로를 통해 가는 거리가 더 짧은지 비교한 후 더 작은 값을 배열에 저장하여 넣어준다.
3. 출력