1/0 Knapsack using Greedy Approach:

Greedy approach does not guarantee the optimal solution for 0/1 knapsack as it chooses items based on immediate benefit without considering the overall weight constraints.

Algorithm:

Select items in decreasing order of their values and include them in the knapsack if they fit, until the knapsack is full.

Code:

```
#include <iostream>
#include <algorithm>
using namespace std;

struct Item {
   int weight, value;
};

bool comparison(Item a, Item b) {
   return (a.value > b.value);
}

int knapsackGreedy(Item items[], int n, int capacity) {
   sort(items, items + n, comparison);
   int totalValue = 0, currentWeight = 0;

   for (int i = 0; i < n; i++) {
      if (currentWeight + items[i].weight <= capacity) {
         currentWeight += items[i].weight;
         totalValue += items[i].value;
      }
   }
   return totalValue;
}

int main() {
   Item items[] = {{10, 60}, {20, 100}, {30, 120}};
   int capacity = 50;
   int n = sizeof(items) / sizeof(items[0]);
   cout << "Maximum value in Knapsack: " << knapsackGreedy(items, n, capacity);
   return 0;
}
```

Output:

```
Maximum value in Knapsack: 220
```

Space complexity: _____

Justification:_____
_____

Time complexity:

Best case time complexity: _____

Justification:_____
_____

Worst case time complexity: _____

Justification:_____