Fractional Knapsack using Greedy Approach:

Fractional knapsack allows taking fractional parts of items to maximize the total value. Greedy approach selects items with the maximum value-to-weight ratio first.

Algorithm:

1. Calculate value-to-weight ratios for all items.
2. Sort items based on ratios in descending order.
3. Take items in order until the knapsack is full, including fractional parts if necessary.

Code:

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

struct Item {
   int weight, value;
   double ratio;
};

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

double fractionalKnapsackGreedy(Item items[], int n, int capacity) {
   for (int i = 0; i < n; i++) {
      items[i].ratio = (double)items[i].value / items[i].weight;
   }
   sort(items, items + n, comparison);

   double totalValue = 0.0;
   int currentWeight = 0;

   for (int i = 0; i < n; i++) {
      if (currentWeight + items[i].weight <= capacity) {
         currentWeight += items[i].weight;
         totalValue += items[i].value;
      } else {
         int remainingWeight = capacity - currentWeight;
         totalValue += items[i].ratio * remainingWeight;
         break;
      }
   }
   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: " << fractionalKnapsackGreedy(items, n, capacity);
    return 0;
}
```

Output:

```
Maximum value in Knapsack: 240
```

Space complexity: _____

Justification:_____
_____

Time complexity:

Best case time complexity: _____

Justification:_____
_____

Worst case time complexity: _____

Justification:_____