CS1110 Lab 10 Solutions

1. Benchmarking Merge

With default params m=1 and p=3 we have:

```
Fujuns-MacBook-Air:Lab_10 fujun$ python BenchMerge.py

Times for Merge1 with n = 1000
   0.00214
   0.00128
   0.00115

Times for Merge2 with n = 1000
   0.00062
   0.00073
   0.00058
Fujuns-MacBook-Air:Lab_10 fujun$ python BenchMerge.py

Times for Merge1 with n = 10000
   0.05850
   0.08394
   0.13093

Times for Merge2 with n = 10000
   0.01740
   0.00769
   0.00977
Fujuns-MacBook-Air:Lab_10 fujun$ python BenchMerge.py

Times for Merge1 with n = 100000
   4.53456
   4.61413
   4.63234

Times for Merge2 with n = 100000
   0.09216
   0.09367
   0.09306
```

From this experiment, Merge2 is always more efficient than Merge1. With larger n, the gap between them also increases, which indicates that “pop” operation is expensive.

2. Comparing Selection Sort and Merge Sort

With params m=1 and p=1 we have:

```
Fujuns-MacBook-Air:Lab_10 fujun$ python BenchSort.py

Times for SelectionSort with n = 1000
   0.03515

Times for MergeSort with n = 1000
   0.00482
Fujuns-MacBook-Air:Lab_10 fujun$ python BenchSort.py

Times for SelectionSort with n = 5000
   0.73906

Times for MergeSort with n = 5000
   0.03003
Fujuns-MacBook-Air:Lab_10 fujun$ python BenchSort.py

Times for SelectionSort with n = 10000
   2.76456

Times for MergeSort with n = 10000
   0.07105
Fujuns-MacBook-Air:Lab_10 fujun$ python BenchSort.py

Times for SelectionSort with n = 20000
  11.33780

Times for MergeSort with n = 20000
   0.12879
Fujuns-MacBook-Air:Lab_10 fujun$ python BenchSort.py

Times for SelectionSort with n = 40000
  53.82041

Times for MergeSort with n = 40000
   0.33900
```

From this experiment, MergeSort is always more efficient than SelectionSort. With larger n, the gap between them also increases.

An intuitive explanation for this would be that, given an array with n items to be sorted, selection sort needs a nested for loop which gives a bigO(n^2) cost, while merge sort repeats subdividing the current array size by 2 (with a linear cost for the current array) which gives a bigO(nlog(n)) cost.

3. ShowTriPartition

(a) How many yellow triangles in a level-L partitioning?

Level	Num
0	1
1	3
2	9
3	27

Each level's partitioning will replace the current yellow with 3 smaller yellow triangles (and a pink triangle), so the number is (3^L).

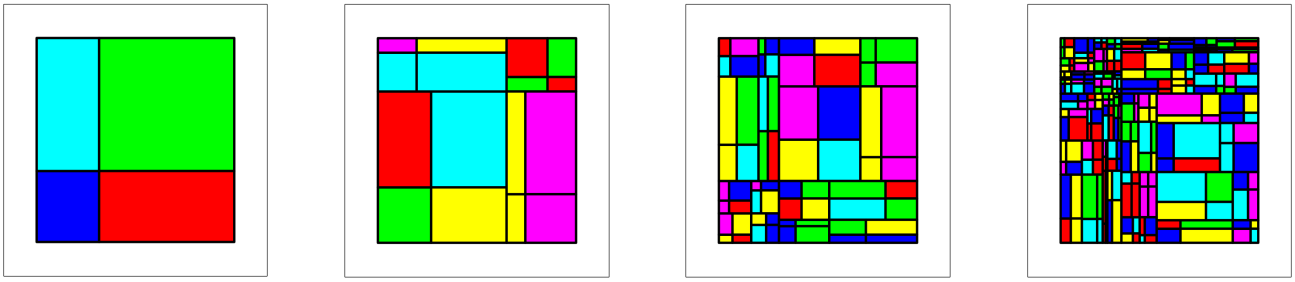
(b) Show how two calls to Partition can produce this graphic:

```
MakeWindow(6,bgcolor=WHITE,labels=True)
x1 = [-5., 5., 5.]
y1 = [-5., -5., 5.]
x2 = [-5., 5., -5.]
y2 = [-5., 5., 5.]
Partition(x1, y1, 3)
Partition(x2, y2, 3)
ShowWindow()
```

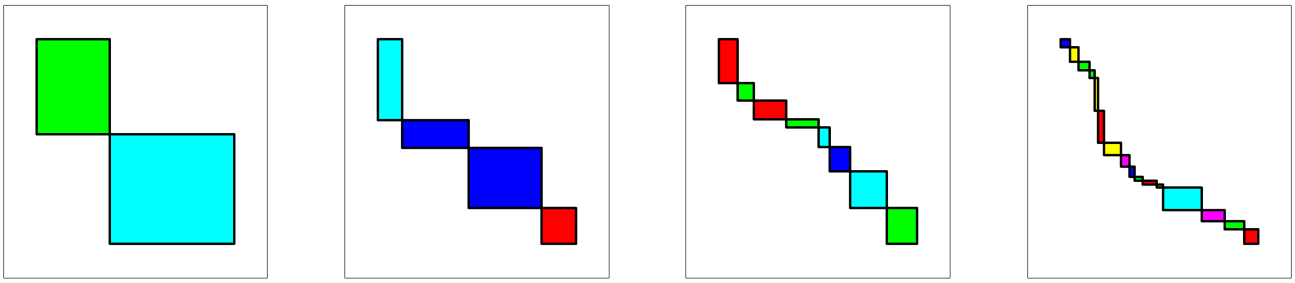
(Note that you need to use floating number "5." instead of "5")

4. Mondrian

Add two new colors: {GREEN, YELLOW}:



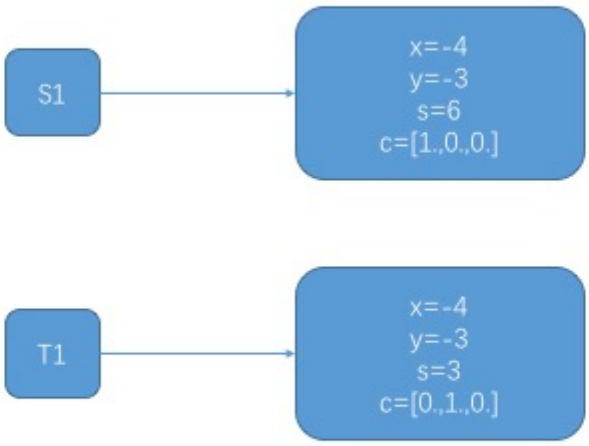
Remove “northeast” and “southwest” subrectangles:



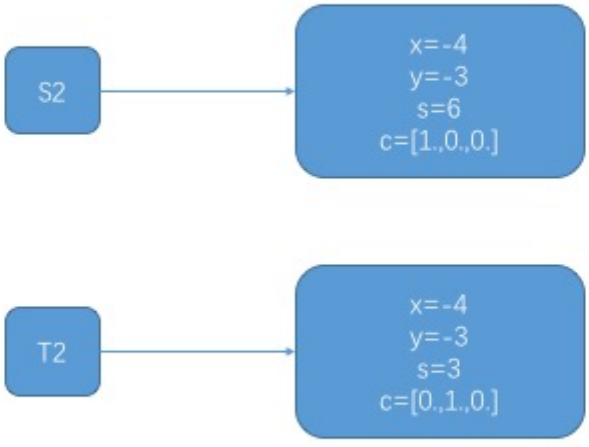
The original implementation will subdivide each rectangle into four smaller rectangles in “northwest”, “northeast”, “southwest” and “southeast” directions. After removing “northeast” and “southwest” directions, in each recursion it won’t draw the “northeast” and “southwest” rectangles, thus formulates a “rectangle chain” from “northwest” to “southeast”.

5. Copying Objects

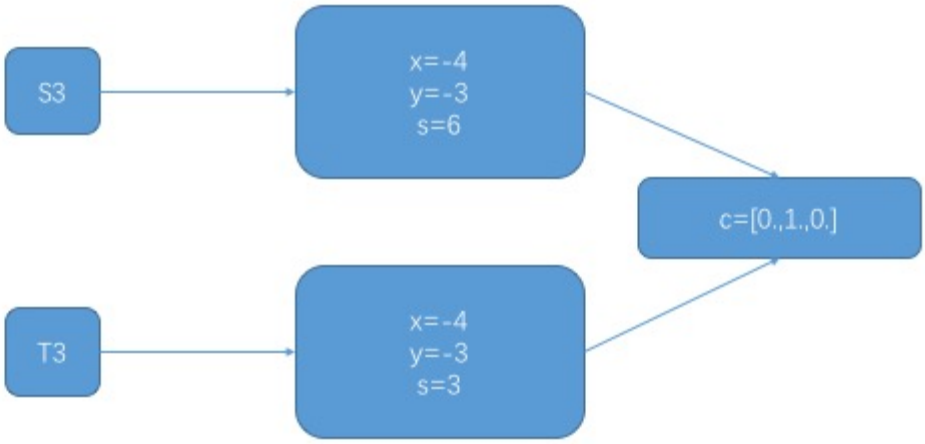
(a)



(b)



(c)



(d)