CS 1301 Exam 3 Study Guide

Pixel Manipulation

So, there are many simple functions to GET the picture you need!

picYay=takePicture() allows the robot to take a picture and save it to p

getThePic=makePicture(picFileName) retrieves the picture saved to your computer and the parameter is the name of the file (as a string) that you want to get (don't forget the .jpg/.gif/etc extension)!

Hokay, so now we have our picture (either we took it or retrieved it), so how do we see it? We show it, *duh.*

show(picName) shows us the picture we took or downloaded!

Two functions that could come in handy to find the height and width:

w=getWidth(picName) stores the width of the picture to w

h=getHeight(picName) stores the height of the picture to h

So, we got da picture and some info on it…to obtain the info we need to manipulate the pixels, we need to know how to look at a *SPECIFIC* pixel

yay=getPixel(picName,x_coord,y_coord) stores the "info" at that coordinate of a pixel…do NOT assume yay has the rgb values or is a list holding all the magical info you need…you have to use other functions to access that! See, look if you don't believe me!

```
1 from Myro import *
2 init("sim")
3 pic=takePicture()
4 yay=getPixel(pic,getWidth(pic)/2,getHeight(pic)/2)
5 #my x and y coord look at the center!
6 print(yay)
```

Output

```
You are using:
   Simulated Fluke, version 1.0.0
   Simulated Scribbler 2, version 1.0.0
Hello, my name is 'Scribby'!
<Graphics+Pixel object at 0x000000000000002B [Graphics+Pixel]>
```

Notice how it printed some random useless info…nothing about RGB y'all!!

tonsOPixels=getPixels(picName) stores "info" about all the pixels…once again, this does not hold the magical info you need! Look!

```
1 from Myro import *
2 init("sim")
3 pic1=takePicture()
4 lotsOPixels=getPixels(pic1)
5 print(lotsOPixels)
```

Output

```
Done
Running '/Users/samie514/Documents/yay.py'...
You are using:
   Simulated Fluke, version 1.0.0
   Simulated Scribbler 2, version 1.0.0
Hello, my name is 'Scribby'!
<Graphics+<getPixels>c__Iterator0 object at 0x000000000000002C [Graphics+<getPixels>c__Iterator0]>
```

Since we are not looking at a specific pixel, what do we do? WE ITERATE THROUGH THEM

```
1 from Myro import *
2 init("sim")
3 pic1=takePicture()
4 lotsOPixels=getPixels(pic1)
5 listOPixels=[]
6 for i in lotsOPixels:
7     listOPixels.append(i)
8 print(listOPixels[0]) #just to show one as an example
```

Output

```
You are using:
   Simulated Fluke, version 1.0.0
   Simulated Scribbler 2, version 1.0.0
Hello, my name is 'Scribby'!
<Graphics+Pixel object at 0x000000000000002D [Graphics+Pixel]>
Done
```

Now we "accessed" the "info" at a certain pixel and each item in listOPixels holds that info!

Now the fun part… COLORS

If you passed kindergarten, you know RGB stands for red, green, and blue. The values range from 0 to 255.

x=getRGB(specificPixel) stores a tuple of the red, green, and blue values to x

r=getRed(specificPixel) stores the red value to r

g=getGreen(specificPixel) stores the green value to g

b=getBlue(specificPixel) stores the blue value to b

Look at the next page to show the example!

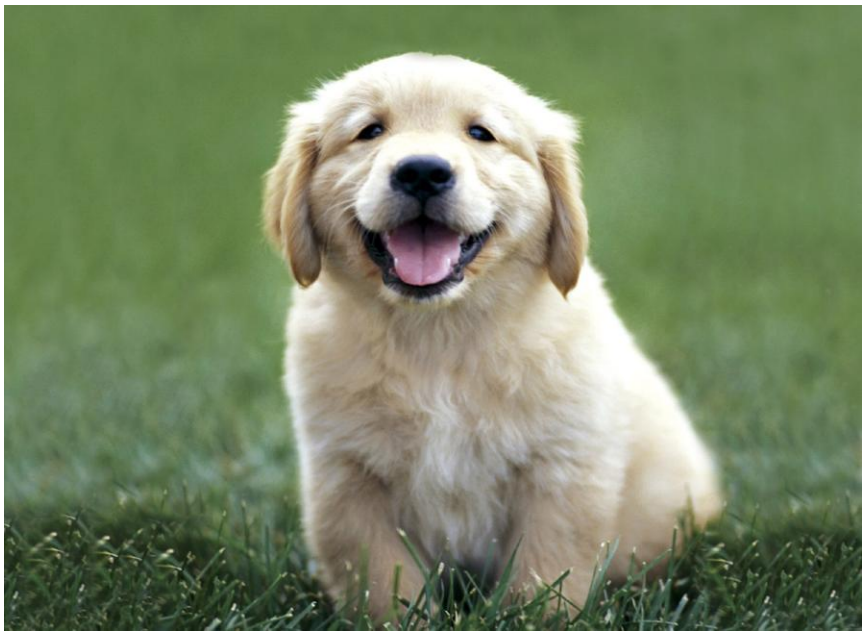
```
1  from Myro import *
2  init("sim")
3  pic1=takePicture()
4  specificPixel=getPixel(pic1,getWidth(pic1)/2,getHeight(pic1)/2)
5  x=getRGB(specificPixel)
6  print(x)
7  r=getRed(specificPixel)
8  print(r)
9  g=getGreen(specificPixel)
10 print(g)
11 b=getBlue(specificPixel)
12 print(b)
```

Output

```
You are using:
   Simulated Fluke, version 1.0.0
   Simulated Scribbler 2, version 1.0.0
Hello, my name is 'Scribby'!
(99, 0, 99)
99
0
99
Done
```

YAYAY TA DA LOOK WE KNOW THE RGB VALUES AFTER ALL OUR HARD WORK

Meanwhile, here is a picture of a puppy:



Sorry, I don't have my gel pens, so I have to distract y'all somehow

So, now that we know how to get the RGB values, what do we do with the info? What is the point of pixel manipulation? Well, some other useful functions…

setRed(specificPixel,hey) sets the specific pixel's red value to hey

setGreen(specificPixel,hi) sets the specific pixel's green value to hi

setBlue(specificPixel,chica) sets the specific pixel's blue value to chica

So, I finally taught y'all the basics of pixel manipulation, but I will leave you guys with some **~fun~** other functions

x_coor=getX(specificPixel) stores the x coordinate to x_coor of specificPixel

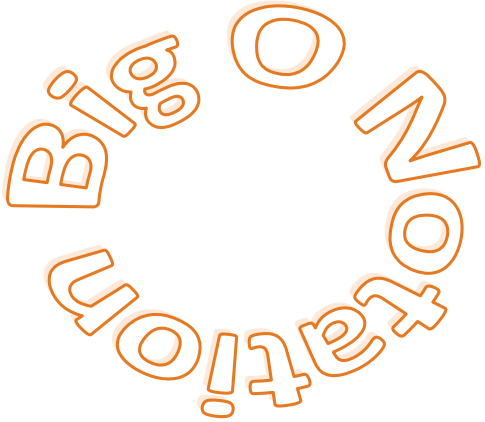
y_coor=getY(specificPixel) stores the x coordinate to y_coor of specificPixel

Okay now that you guys have learned the functions to create your beautiful artwork, I am sure you want to save it, right?

savePicture(pic, whatToSaveAs) saves pic as whatToSaveAs (string, include extension y'all)

K, so your picture was sOoOoOoOoOooooo cute, you want a copy!

copyCat=copyPicture(pic) copies pic and stores to copyCat



So what the h%&^ is big O notation? What does this entitle? Why are we learning this? WHY DOES CS HATE ME?

….Okay maybe that was too much. Point being, it isn't too bad if you think about it logically. Big O teaches us how an algorithms work grows as the input increases. As input increases, different algorithms can grow at different paces. (Tip: when trying to figure out which Big O it is, ignore constants!)

We have Big O's:

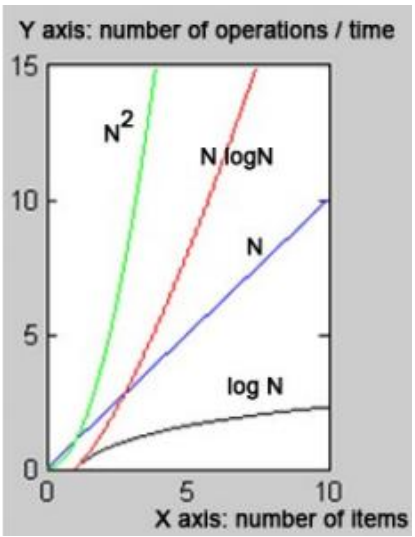
-> N

-> log(N)

-> N^2

-> Nlog(N)

We will be utilizing Big O more when we go into sorting and searching!For now, review your graphs of the above Big O's…or just look at this one, lulz



Searches and Sorting!

We got dem searches and we got dem sortin so…let's start with searches!

Searches

Linear Search-

This searches old sk00l aka one by one (literally)

So linear would imply what Big O? N!

Binary Search-

Compare the target value to the midpoint of the list. If it's not the target, see if the midpoint is the higher or lower than the target. Divide in half and choose the correct half. Repeat until you run out, or you find the target (yippee!). THIS SEARCH. CAN. ONLY. BE. PERFORMED. ON. SORTED. LISTS. TAKE. NOTE. OF. MY. EMPHASIS.

Example time!

Search 2 in list [1, 2, 3, 5, 8, 10, 15, 25]

Round 1: Mid point: 8

2<8

New list: [1, 2, 3, 5]

Round 2: Mid point: 3

2<3

New list: [1, 2]

Round 3: Mid point: 2

2=2

YAYYYYYY

So what is our Big O? logN

Sorting

Selection Sort-

Select the smallest number (if sort increasingly) in the list and append it to the result list.

Example time!

Sort [3, 1, 4, 2] increasingly

Round 1: Minimum: 1

Result list: [1]

New list: [3, 4, 2]

Round 2: Minimum: 2

Result list: [1, 2]

New list: [3, 4]

Round 3: Minimum: 3

Result list: [1, 2, 3]

New list: [4]

Round 4:Minimum: 4

Result list: [1, 2, 3, 4]

So what is our Big O? N^2 (N rounds. At most N comparisons each round tofind out the smallest element.)

Insertion Sort:

Get the first element in the list. Insert it in the right place in

the result list.

Example time!

Sort [3, 1, 4, 2] increasingly

Round 1:

Element: 3

Result list: [3]

New list: [1, 4, 2]

Round 2:

Element: 1

Result list: [1, 3]

New list: [4, 2]

Round 3:

Element: 4

Result list: [1, 3, 4]

New list: [2]

Round 4:

Element: 2

Result list: [1, 2, 3, 4]

YAYAYAYAYAYAYAYAYAYAYAY Y'ALL

What's our BigO? N^2(N rounds. At most N comparisons each round tofind out the correct location.)

Bubble Sort:

Pass through the list of elements, and swap adjacent elements if they are not in the correct order. It must repeat the pass N-1 times to guarantee the entire list is sorted (If the smallest element is at the end of the list, it will take N-1 passes to swap it down to the front of the list.)

Example time!

Sort [3, 1, 4, 2] increasingly

Round 1:

[3, 1, 4, 2] →[1, 3, 4, 2]

[1, 3, 4, 2]→ [1, 3, 4, 2]

[1, 3, 4, 2] →[1, 3, 2, **4**]

The last element in the list is guaranteed to be correct after the first round.

Round 2:

[1, 3, 2, **4**] →[1, 3, 2, **4**]

[1, 3, 2, 4] →[1, 2, **3, 4**]

The last two elements in the list is guaranteed to be correct after the second round.

Round 3:

[1, 2, **3, 4**] →[1, **2, 3, 4**]

The last three elements in the list is guaranteed to be correct after the third round.

TAAA DA!

So what is our BigO? N^2

Merge Sort:

Divide the original list into small lists. Merge the small lists.

Example time!

Sort [3, 1, 4, 2] increasingly

Division stage:

Round 1:

[3, 1, 4, 2] →[3, 1] [4, 2]

Round 2:

[3, 1] [4, 2]→ [3] [1] [4] [2]

Merge stage: We broke it all up, now we gotta put all numbers back into one list!

Round 1:

[3] [1] [4] [2]→ [1, 3] [2, 4]

Round 2:

[1, 3] [2, 4] → [1, 2, 3, 4]

hehe ☺

What's our BigO? NlogN

Quick Sort:

Select element as pivot every round and compare the rest elements to the pivot. Elements that are less than the pivot are collected into an unsorted list on the left of the pivot. Elements that are greater than or equal to the pivot are collected into an unsorted list to the right of the pivot. Repeat for the left and right hand collection of numbers until the size of each collection is one, at which point the entire list of numbers is correctly ordered.

Example time!

Sort [3, 1, 4, 2] increasingly

Round 1:

Pivot: 4 (random choice)

New list: [3, 1, 2, **4**]

Round 2:

Pivot: 1 (random choice)

New list: [**1**, 3, 2, **4**]

Round 3:

Pivot: 2 (random choice)

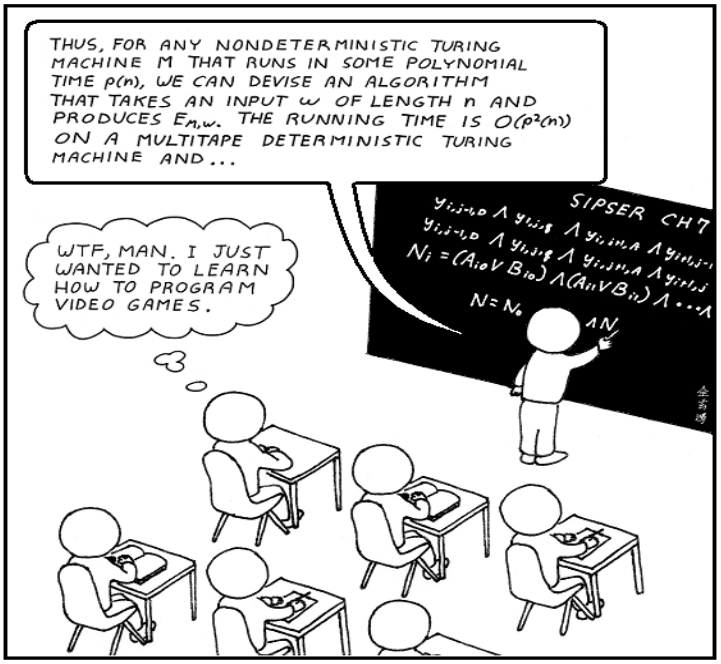
New list: [**1**, **2**, 3, **4**]

So about BigO… It depends on which pivot you choose!

→Average is NlogN

→Maximum is N^2

All of you after reading about sorting:



And here is a joke for you guys (explanation to follow):

A little girl goes into a pet show and asks for a wabbit. The shop keeper looks down at her, smiles and says:
“Would you like a lovely fluffy little white rabbit, or a cutesy wootesly little brown rabbit?”
“Actually”, says the little girl, “I don’t think my python would notice.”
HAHAHAHAHAHAHAHAHA GET IT? HE USED AN OR OPERATOR SO EITHER WORKS (please laugh) (no, really)

Okay, BACK TO WORK ON RECURSION.

So, what is recursion? Recursion is calling a function that is currently executing. Recursion has THREE parts that you NEED TO KNOW!
They are…

-Base case aka terminating condition…if you do not tell the recursion when to stop, it will be infinite!

-Recursion call- This is when you call the function within the function (simple enough, eh?)

-Recursive step- Usually using an increment or decrement, it will help you reach the base case!

Example:

```
1 #Computes x raised to power of n
2 def exp(x, n):
3     if n == 0: #base case
4         return 1
5     else:
6         return x * exp(x, n-1)#our recursion call and recursive step
7 print(exp(3,4))
```

Output

```
Running '/Users/samie514/Documents/yay.py'...
81
Done
```

```
exp(2, 4)
+-- 2 * exp(2, 3)
|       +-- 2 * exp(2, 2)
|       |       +-- 2 * exp(2, 1)
|       |       |       +-- 2 * exp(2, 0)
|       |       |       |       +-- 1
|       |       |       +-- 2 * 1
|       |       |       +-- 2
|       |       +-- 2 * 2
|       |       +-- 4
|       +-- 2 * 4
|       +-- 8
+-- 2 * 8
+-- 16
```

Tips for the exam!

- As usual, know your vocab!
- Review over old material…everything builds on each other!
- MAKE SURE YOU REALIZE getPixel(pic,x,y) AND getPixels(pic) DO NOT POINT AT ALL THE MAGICAL INFO IN THE WORLD
- I know recursion can be tricky, but practicing helps! Also, draw diagrams on a big whiteboard. You have tons of space to literally put your logic on board!
- Come up with some example photo effects! Test them on your own photos. Put a rectangle on a friend's head, or draw a horizontal line mustache on them (then post it on their wall for lolz)
- Know your old exams. If you did badly on (for example) the second exam, don't go all YOLO on me and wing the third based off solely the new material. File I/O, lists, tuples, etc WILL come back to haunt you.
- As I ALWAYS SAY, ask for help. There is no such thing as a stupid question unless you say, "Can you give us a test question?" because HA NO.

Email me!

skassem3@gatech.edu

☺