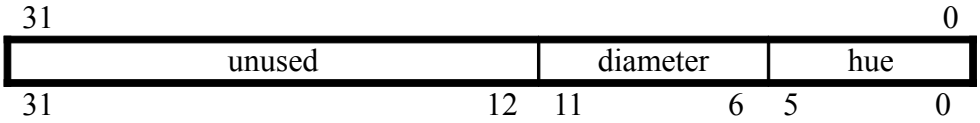
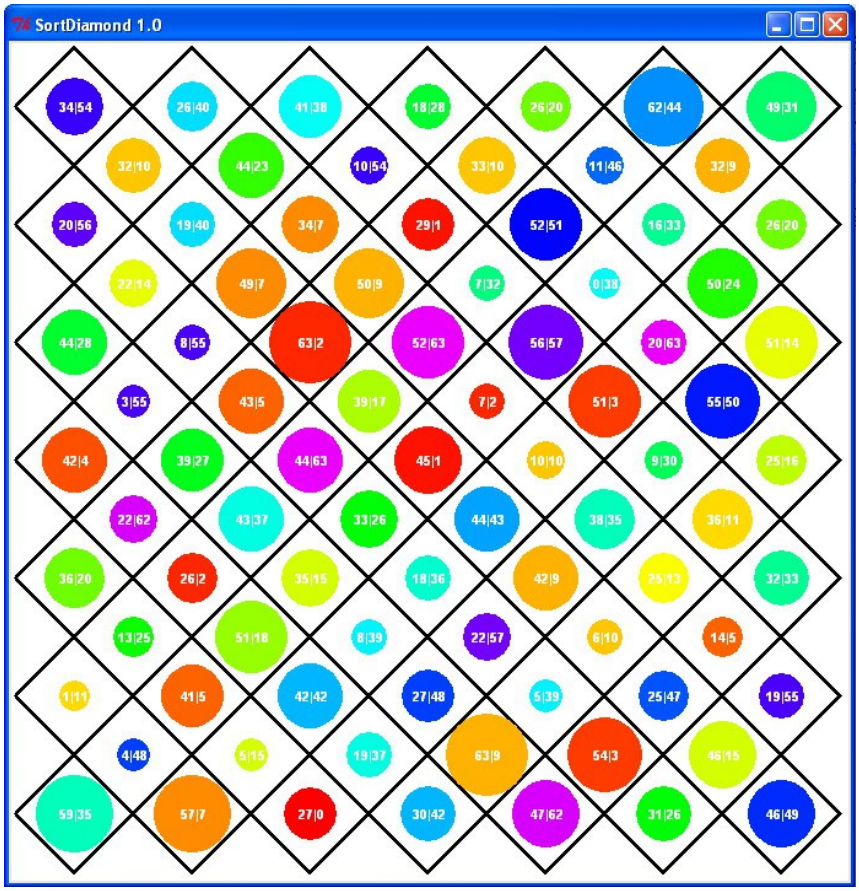
Part 1: This part explores the development and implementation of a complex algorithm to perform a two parameter orthogonal sort with spatial coupling. The program begins with a 13 x 13 sort diamond containing 85 disks of varying diameter and hue. A rearrangement must be derived where each row includes *monotonically increasing hue* (color hue) from left to right and each column includes *monotonically increasing diameter* from top to bottom. An example diamond is shown below. In order to minimize screen usage, the diamond has been rotated counterclockwise by 45°. So hue increases from the bottom and left edges to the upper and right edges. Diameter should increase from the top and left edges to the bottom and right edges.

The DiamondSort data is placed in memory through a provided routine. After initialization, the diameter and hue are stored in static memory in a 85 elements in a 169 linear array using a packed format, where missing elements are represented as -1. The least significant six bits represents one of 64 hues (0-63) and the next least significant six bits represents diameter (0-63). Actual displayed diameters are increased by 20 to improve visibility and hue identification.



The data is placed in a 13 x 13 square that subscribes the upright diamond. All unused array entries are initialized to -1. Since storage will be a measured cost in the performance version of this program (Part 2), it is strongly recommended that your routine exchange values rather then creating a new array.



When your implementation completes, the board should contain the same tiles in the original board, only sorted by rows (hue) and columns (diameter). The shell program `P1-1-shell.c` includes a reader function `Load_Mem()` that loads the values from a text file. The shell program also includes a printing function to display the final sorted diamond board. You should use `gcc` under Ubuntu to develop your program (type `man gcc` for compiler usage). Sample value files `sortboard1.txt, sortboard2.txt, and sortboard3.txt` are provided. Normally, you should compile and run your program using the Linux command line:

```
> gcc P1-1.c –g –Wall –o P1-1
> ./P1-1 sortboard1.txt
```

1. The file must be named `P1-1.c`.
2. Your name and the date should be included in the beginning of the file. Your program should display no warnings or errors when compiled and executed using gcc.
3. Your program should not modify the data in the input file; rather it should rearrange the data within the array and then print out the data using the included diamond print function Print_Array().
4. Your solution to this part (P1-1) is due **Friday, 17 June 2016 at 5:00 PM**. You have a grace period for submission until 11:55 PM. Your submission will be marked late, but no penalty will be assessed. After 11:55 PM, submissions will not be accepted.

Part 2: In this part, a performance version of DiamondSort is developed in MIPS assembly. In this version, the code size, execution time, and storage will contribute to the overall evaluation of the submitted code. The choice of representation plays a critical role in the final implementation.

Library Routines: There are three library routines (accessible via the `swi` instruction) for use in this project.

SWI 521: Diamond Sort Initialization: This routine creates a diamond subscribed in a 13 x 13 array in memory. 85 packed values representing a hue and diameter are positioned within a 169 element (13 x 13) integer array. A pop up window also graphically represents the initial contents of the sort board. INPUTS: $1 contains the base address of the square array. OUTPUTS: none.

SWI 522: Diamond Sort Checking: This routine checks a diamond beginning at the specified base address. The number of decreasing hues is tallied in rows and the number of decreasing diameters is tallied in columns. The two values are returned as two packed six-bit integers in $2. INPUTS: $1 contains the base address of the square array. OUTPUTS: $2 contains the number of row errors (bits 5-0) and column errors (bits 11-6).

SWI 523: Diamond Sort Display Update: This routine updates the pop up window with the current contents of the diamond in memory. INPUTS: $1 contains the base address of the square array. OUTPUTS: none.

An assembly language shell code is provided (`P1-2-shell.asm`).

Evaluation: In this version, correct operation and efficient performance are evaluated. The code size, dynamic execution length, and operand storage requirements are scored empirically, relative to a baseline solution. The baseline numbers for this project are static code size: 66 instructions, dynamic instruction length: 13,910 instructions (avg.), storage required: 100 words (including the 85 words in the diamond but excluding dedicated registers $0, $31). Your score will be determined through the following equation:

$$PercentCredit = 2 - \frac{Metric_{YourProgram}}{Metric_{BaselineProgram}}$$

Percent Credit is then used to determine the number of points for the corresponding points category. Important note: while the total score for each part can exceed 100%, especially bad performance can earn *negative credit*. The sum of the combined performance metrics scores (code size, execution length, and storage) will not be less than zero points. **Finally, the performance scores will be reduced by 10% for each incorrect trial (out of 100 trials). You cannot earn performance credit if your implementation fails ten or more of the 100 trials.**

In order for your solution to be properly received and graded, there are a few requirements.

1. The file must be named `P1-2.asm`.
2. Your name and the date should be included in the beginning of the file. Your program should display no warnings or errors when loaded and executed in MiSaSiM.
3. Your program should rearrange the data within the array so that swi 522 returns zero.
4. Your program must return to the operating system via the **jr** instruction. *Programs that include infinite loops or produce simulator warnings or errors will lose significant credit*.
5. Your solution to this part (P1-2) is due **Wednesday, 29 June 2016 at 5:00 PM**. You have a grace period for submission until 11:55 PM. Your submission will be marked late, but no penalty will be assessed. After 11:55 PM, submissions will not be accepted.

Good luck!

Project Grading: The project grade will be determined as follows:

part	*description*	*percent*
P1-1	Functional DiamondSort	
	Algorithm	5
	Style/comments	5
	Compiles	5
	Runs without crashing	5
	Correctness	20
	total	40
P1-2	Performance DiamondSort	
	correct operation, proper technique and style	20
	static code size	10
	dynamic execution length	20
	operand storage requirements	10
	total	60