CHAPTER 5
IMPLEMENTATION AND TESTING

5.1 Implementation

This application is creates *Counting* and *Radix* sorting algorithms visualization media using *HTML* and *Javascript* programming language. First step while creating this application, data which will input by user were made on *HTML* interface. Main form are need text form for obtaining data from user input, selecting option for animational speed, and submit button which have form action for switch to each another page and starting algorithms visualization.

On *canvas* from each algorithms web-page, this application need method to storing data from user input and sort with choosen sorting algorithms. In *counting* sort, data will be processed first and each result of iteration will be stored in temporary array for further animation process. Meanwhile in *radix* sort, recursive function on sorting process allows calling of animation function in each loop, but this becomes trouble when determining time in pause and resume functions.

Before composing visualization function, *canvas* needs more than one method with doing apart to make an object like a rectangle and text. For example to write a text on *canvas*, some method need to declare first, like a color, size, transparency, align, etc. To make it efficient, application needs function to create an object. Code below explains to create new function to make text object in *canvas*.

```
1. function createText(content,x,y,color,font,align,transparency){
2.   ctx.fillStyle   = color;
3.   ctx.font        = font;
4.   ctx.textAlign   = align;
5.   ctx.globalAlpha = transparency;
6.   ctx.fillText(content,x,y);
7. }
8. createText(“Counting Sort”, 10,10, white, light,left,1);
```

Basically, to make animation of algorithms steps, object in *canvas* is creating on certain interval and timeout. Calculating interval, timeout and position of object were stand as main difficulty on this application. For example, code below is containing function to create and recreate object on certain interval in counting sort. To make this function can be reusable, function need to be declared as variable.

```
1. var countingInitTicker =
2.       function(){
3.       createRect(boxwidth,boxheight,x[4]-xdif+spacera,y[2]-
   ydif,red);
4.       createText(initArray[a],x[4]+spacera,y[2],white,regular,
   center,1.0);
5.       createRect(boxwidth,boxheight/10,x[4]-xdif+spacera,y[2]-
   ydif+boxheight,orange);
6.       for(i=min;i<=max;i++){
7.             if(initArray[a]==i){
8.                   createRect(boxwidth,boxheight,x[4]-
   xdif+xspacer*(i-min),y[3]-ydif,blue);
9.                   createText(counter[i],x[4]+xspacer*(i-
   min),y[3],white,regular,center);
10.                  createRect(boxwidth,boxheight/10,x[4]-
   xdif+xspacer*(i-min),y[3]-ydif+boxheight,orange);
11.                  counter[i]++;
12.            }
13.      }
14.      spacera=spacera+xspacer;
15.      a++;
16.      if (spacera == sizeof*xspacer){
17.            clearInterval(tickcount);
18.            clearRect(boxwidth*sizeof,boxheight/10,x[4]-
   xdif+spacera,y[2]-ydif+boxheight);
19.            tickcount=setInterval(countingResultTicker,delay);
20.      }
21.      }
22.      tickcount = setInterval(countingInitTicker,delay);
```

Actually, interval method is sufficient on creating animation, but it still has limitations in managing recursive objects because interval itself is looping method. To overcome this problem, in making *radix* sort animation, timeout method is needed to do better looping method which can be set more easily. Piece of code below is using interval inside timeout method to create a radix sort animation.

```
1. function radixVisualizer(array2,count2){
2.       var radix=  function(){
3.             radixObject();
4.             //radixTicker function has setInterval.
5.             radixTicker(array2,count2,blue);
6.       }
7. }
8. timer = setTimeout (radix,delay*(sizeof+1)*count2);
```

Although only using *canvas*, visualization of algorithms creates as close as possible with an actual sorting process. Representation of object in certain time and condition can makes animation happen in each data movement.

5.2 Testing

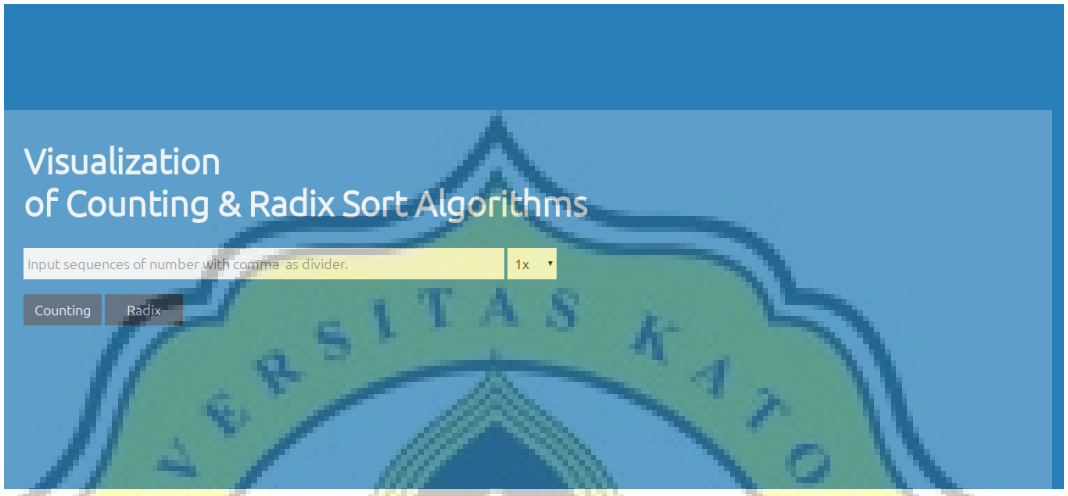

Illustration 5.1: Index Page

On starter page, application requires user input with form of a sequence of numbers with "," (comma) as divider. In same form, user also can choosing animation speed. To start visualization on each other page, user choosing sorting algorithms by pressing button of *counting* or *radix* sorts.

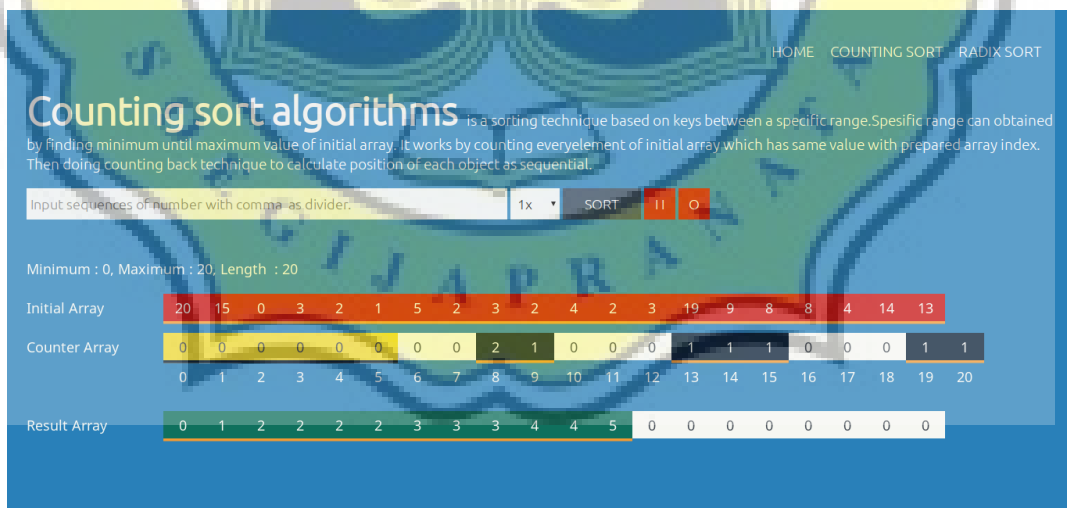

Illustration 5.2: Counting Sort Page

Counting page has a form of input value to be visualized by *counting* sort. *Canvas* are containing array with initial values, array for counter with length of array obtained from range between minimum to maximum values from initial

values, and of course displaying result from sorting process. Limitation from this page is *canvas* just can handle less than 20 for data amount and range of data.

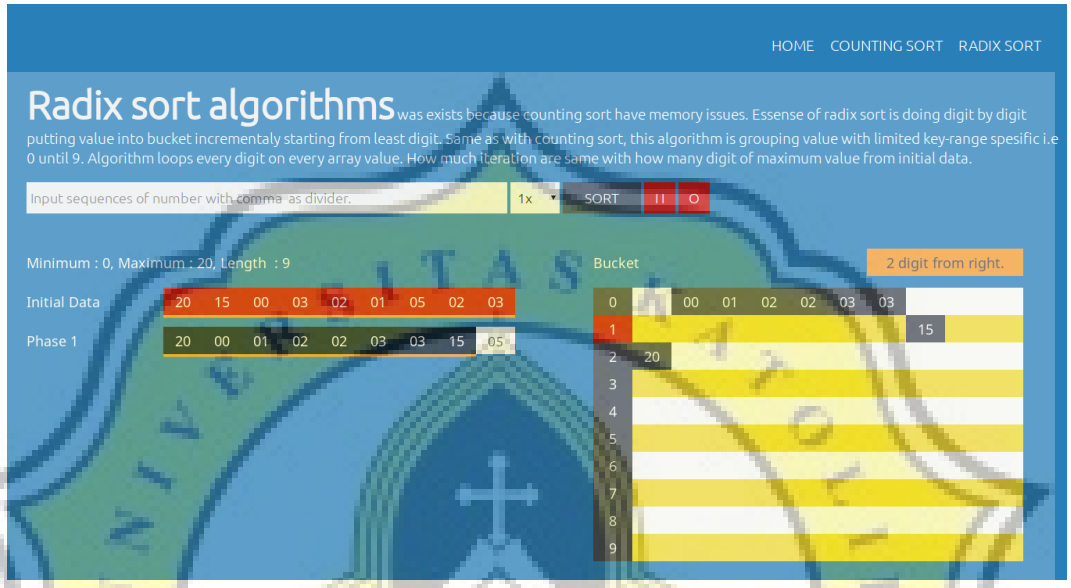

Illustration 5.3: Radix Sort Page

On *Radix* page, user can input values to visualize with *radix* sort. *Canvas* on this page presents initial array values, each iteration arrays, and result array. Arrays data from each iteration of *radix* sort algorithms are passes onto bucket for grouping by every digit. Same as counting sort, these *canvas* space is not enough while handling data in some conditions. That conditions are data amount more than 10 and data length more than 4 digit.

After testing entire application, error about stopping animation function was found. Using interval inside timeout method actually can shortening code, but causing problem. It happens because visualization function is called along with algorithms function of *radix* sort recursively. Coupled with it, modifify data basically is impossible if *canvas* only using interval and timeout while running animation. To overcome this problem, entire visualization from this project need to recreates using animation frame from javascript or using jquery to controlling entire animation.