```
\documentclass{article}
\usepackage{graphicx} % Required for inserting images
\usepackage{amsmath}
\usepackage{calc}
\usepackage{eso-pic}

\newlength{\PageFrameTopMargin}
\newlength{\PageFrameBottomMargin}
\newlength{\PageFrameLeftMargin}
\newlength{\PageFrameRightMargin}

\setlength{\PageFrameTopMargin}{1cm}
\setlength{\PageFrameBottomMargin}{1cm}
\setlength{\PageFrameLeftMargin}{1cm}
\setlength{\PageFrameRightMargin}{1cm}

\makeatletter

\newlength{\Page@FrameHeight}
\newlength{\Page@FrameWidth}

\AddToShipoutPicture{
  \thinlines
  \setlength{\Page@FrameHeight}{\paperheight-\PageFrameTopMargin-\PageFrameBottomMargin}
  \setlength{\Page@FrameWidth}{\paperwidth-\PageFrameLeftMargin-\PageFrameRightMargin}
  \put(\strip@pt\PageFrameLeftMargin,\strip@pt\PageFrameTopMargin){
    \framebox(\strip@pt\Page@FrameWidth, \strip@pt\Page@FrameHeight){}}}

\makeatother
\color{black}
```

```latex
\title{\fontsize{16pt}{20pt}\selectfont{\textbf{\textrm{\underline{Experiment No: 14 }}}}}

\date{}

\begin{document}

\maketitle

\begin{itemize}

\item{\fontsize{14pt}{20pt}\textbf{{\underline{Title:} Insertion Sort and Selection Sort}}

\end{itemize} \\

\begin{itemize}

\item{\fontsize{12pt}{20pt}\textbf{\underline{AIM:}Implement the sorting algorithm:Insertion and
Selection Sort }}

\end{itemize}\\

\begin{itemize}

\item{\fontsize{12pt}{20pt}\textbf{\underline{Learning Objective:}}}\end{itemize}

  Students should able to design and analyze simple linear and non linear data structures.

It strengthen the ability to the students to identify and apply the suitable data structure for the given
real world problem.

It enables them to gain knowledge in practical applications of data structures .

\section*{\fontsize{12pt}{20pt}\textbf{{Introduction to sorting algorithms:}}}

{\fontsize{12pt}{20pt}Sorting algorithms are a set of instructions that take an array or list as an input
and arrange the items into a particular order.
```

Sorts are most commonly in numerical or a form of alphabetical (or lexicographical) order, and can be in ascending (A-Z, 0-9) or descending (Z-A, 9-0) order.

Since they can often reduce the complexity of a problem, sorting algorithms are very important in computer science. These algorithms have direct applications in searching algorithms, database algorithms, divide and conquer methods, data structure algorithms, and many more.}

\section*{{\fontsize{12pt}{20pt}\textbf{{Some Common Sorting Algorithms:}}}}

{\fontsize{12pt}{20pt}Some of the most common sorting algorithms are:

a) Insertion sort

b) Selection sort}

\begin{itemize}

{\fontsize{12pt}{20pt}\item\textbf{ Insertion sort:}}

\end{itemize}

{\fontsize{12pt}{20pt} Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.}

{\fontsize{12pt}{20pt}\textbf{Example:}}

{\fontsize{12pt}{20pt}In Insertion sort, you compare the key element with the previous elements. If the previous elements are greater than the key element, then you move the previous element to the next position.

Start from index 1 to size of the input array.}

$$\begin{bmatrix}
8 & 3 & 5 & 1 & 4 & 2\\
\end{bmatrix}
$$

{\fontsize{12pt}{20pt}\textbf{Step 1 :}}

{\fontsize{12pt}{20pt}key = 3 //starting from 1st index.

Here `key` will be compared with the previous elements.

In this case, `key` is compared with 8. since 8 > 3, move the element 8 to the next position and insert `key` to the previous position.

Result: [3 8 5 1 4 2]}\\

{\fontsize{12pt}{20pt}\textbf{Step 2 :}}

\includegraphics[scale=0.5]{step2}

{\fontsize{12pt}{20pt} key = 5 //2nd index

$8 >5$ //move 8 to 2nd index and insert 5 to the 1st index.

Result: [3 5 8 1 4 2]}\\

```
{\fontsize{12pt}{20pt}\textbf{Step 3 :}}

\includegraphics[scale=0.5]{step3}

{\fontsize{12pt}{20pt} key = 1 //3rd index

    $ 8 > 1   => $     $\begin{bmatrix}
   3 & 5 & 1 & 8 & 4 & 2\\
\end{bmatrix}
$

     $5 > 1     =>$    $\begin{bmatrix}
   3 & 1 & 5 & 8 & 4 & 2\\
\end{bmatrix}
$

     $3 > 1     =>$     $\begin{bmatrix}
   1 & 3 & 5 & 8 & 4 & 2\\
\end{bmatrix}
$


      Result: [ 1 3 5 8 4 2 ]}\\


{\fontsize{12pt}{20pt}\textbf{Step 4 :}}

\includegraphics[scale=0.5]{step4}

    {\fontsize{12pt}{20pt} key = 4 //4th index
    $ 8 > 4   =>$    $\begin{bmatrix}
   1 & 3 & 5 & 4 & 8 & 2\\
\end{bmatrix}
$

   $  5 > 4   => $     $\begin{bmatrix}
   1 & 3 & 4 & 5 & 8 & 2\\
\end{bmatrix}
$

  $   3 > 4   \neq> $ stop

     Result: [ 1 3 4 5 8 2 ]}\\



   {\fontsize{12pt}{20pt} \textbf{Step 5 :}}

    \includegraphics[scale=0.5]{step5}

    {\fontsize{12pt}{20pt} key = 2 //5th index

    $ 8 > 2   =>$        $\begin{bmatrix}
   1 & 3 & 4 & 5 & 2 & 8\\
\end{bmatrix} $

    $ 5 > 2   => $      $\begin{bmatrix}
   1 & 3 & 4 & 2 & 5 & 8\\
\end{bmatrix}$

   $  4 > 2   => $      $\begin{bmatrix}
   1 & 3 & 2 & 4 & 5 & 8\\
```

\end{bmatrix} $

$ 3 > 2   =>$ $\begin{bmatrix}

1 & 2 & 3 & 4 & 5 & 8\\

\end{bmatrix} $

$ 1 > 2   \neq> $stop

Result: [1 2 3 4 5 8]}\\

\includegraphics[scale=0.5]{laststep.png}

{\fontsize{12pt}{20pt}The algorithm shown below is a slightly optimized version to avoid swapping the key element in every iteration. Here, the key element will be swapped at the end of the iteration (step).

\begin{verbatim}

```
InsertionSort(arr[])
for j = 1  to arr.length
key = arr[j]
i = j - 1
while i > 0 and arr[i] > key
arr[i+1] = arr[i]
i = i - 1
arr[i+1] = key}
```

\end{verbatim}

\begin{itemize}

{\fontsize{12pt}{20pt}\item\textbf{Properties:}}

\end{itemize}

{\fontsize{12pt}{20pt}1)Space Complexity: O(1)

2) Time Complexity: O(n), O(n* n), O(n* n) for Best, Average, Worst cases respectively.

3) Best Case: array is already sorted.

4) Average Case: array is randomly sorted.

5) Worst Case: array is reversely sorted.

6) Sorting In Place: Yes.

7) Stable: Yes.}

\begin{itemize}

{\fontsize{12pt}{20pt}\item\textbf{Example:}

```
\end{itemize}

\begin{verbatim}

#include <stdio.h>

void insertionSort(int arr[], int n) {
  for (int i = 1; i < n; i++) {
   int key = arr[i];
    int j = i - 1;
    while (j >= 0 && arr[j] > key){
     arr[j + 1] = arr[j];
     j--;
   }
    arr[j + 1] = key;
  }
}

int main() {
 int arr[] = {12, 11, 13, 5, 6};
  int n = sizeof(arr) / sizeof(arr[0]);

  insertionSort(arr, n);

  printf("Sorted array: ");
  for (int i = 0; i < n; i++) {
   printf("%d ", arr[i]);
  }
  printf("\n");

  return 0;
```

}

Output:

Sorted array: 5 6 11 12 13

\end{verbatim}

\begin{itemize}

{\fontsize{12pt}{20pt}\item\textbf{ Selection sort:}}

\end{itemize}

{\fontsize{12pt}{20pt}Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list.

To sort a list using selection sort, we start by finding the smallest element in the list and swapping it with the first element. Then, we find the smallest element in the remaining unsorted portion of the list and swap it with the second element. We continue this process until the entire list is sorted.

Selection Sort is one of the simplest sorting algorithms. This algorithm gets its name from the way it iterates through the array: it selects the current smallest element, and swaps it into place.

Here's how it works:

Step 1)Find the smallest element in the array and swap it with the first element.

\includegraphics[scale=0.5]{S1.png}

Step 2)Find the second smallest element and swap with with the second element in the array.

\includegraphics[scale=0.5]{S2.png}

Step 3)Find the third smallest element and swap wit with the third element in the array.

\includegraphics[scale=0.5]{S3.png}

Step 4)Repeat the process of finding the next smallest element and swapping it into the correct position until the entire array is sorted.

\includegraphics[scale=0.5]{S4.png}

Step 5) Result\$:\$

\includegraphics[scale=0.5]{S5.png}

But, how would you write the code for finding the index of the second smallest value in an array?

An easy way is to notice that the smallest value has already been swapped into index 0, so the problem reduces to finding the smallest element in the array starting at index 1.} \\\\\\\\\\

\begin{itemize}

{\fontsize{12pt}{20pt}\item\textbf{Example:}

\end{itemize}

\begin{verbatim}

```
#include <stdio.h>
```

```
void selection_sort(int arr[], int n)

{
  int i, j, min_idx;

  for (i = 0; i < n - 1; i++) {
    min_idx = i;

    for (j = i + 1; j < n; j++) {
      if (arr[j] < arr[min_idx]) {
        min_idx = j;
      }
    }

    int temp = arr[i];
    arr[i] = arr[min_idx];
    arr[min_idx] = temp;
  }
}

// Function to print the array
void print_array(int arr[], int n) {
  int i;
  for (i = 0; i < n; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
}

int main() {
  int arr[] = {6, 3, 8, 12, 9};
  int n = sizeof(arr) / sizeof(arr[0]);


  printf("Unsorted array: ");
  print_array(arr, n);


  selection_sort(arr, n);


  printf("Sorted array: ");
  print_array(arr, n);


  return


0;
}
```

Output:

```
Unsorted array: 6 3 8 12 9
Sorted array: 3 6 8 9 12
```

\end{verbatim} \newpage

{\fontsize{12pt}{20pt}\item\textbf{\underline{Lab Outcome:}}} \\\\

1) Write functions to implement linear and non-linear data structure operations.\\\\

2) Suggest appropriate linear / non-linear data structure operations for solving a given problem

\end{document}