

Fase 5. Análisis de cada idea propuesta, resaltando los Pros y Contra de cada solución.

Ventajas Y Desventajas:

Alternativa 1:

Esta alternativa tiene una ventaja principal que tanto muestra los datos principales en el que la aplicación funciona y expone la funcionalidad necesaria para procesar los datos a través de interfaz de usuario, y dependiendo de la entrada que el usuario ingrese, se habilitaran mecanismos como barras de menús, barras de herramientas, etc. Esta alternativa a la hora de ingresar los datos es muy buena ya que por medio de los cuadros emergentes mostrara información específica a los usuarios, recopilar a información de los usuarios, y también puede mostrar y recopilar información. Una desventaja de usar un cuadro emergente es que si la entrada que va a ingresar el usuario no será mostrada correctamente dentro del cuadro emergente. Por otra parte, una ventaja de usar el texto plano como mecanismo de ingreso de datos es que el usuario no tendrá que invertir tiempo a la hora de poner los elementos a ordenar, ya que estos estarán previamente listos en el archivo de texto plano. Una desventaja del archivo de texto plano es que tomara un tiempo estipulado a la hora de leer el archivo. Puede haber errores de lectura que el usuario por no tener conocimiento de cómo deberían de ser ingresados los datos no pueda avanzar.

Una ventaja de usar los métodos de ordenamiento, Merge Sort, Selection Sort, y Shell Sort es que estos tienen una complejidad de tiempo mucho menor en comparación con otros algoritmos de ordenamiento, lo que lleva a ordenar los elementos en el menor tiempo posible. Una de las desventajas que podemos encontrar en algunos de esos algoritmos es la implementación en la fase del mundo, ya que pueden llegar a ser un poco complejos de procesar en el sistema. A la hora de la salida los elementos serán mostrados por medio de un JList. La ventaja de este mecanismo es que nos permitirá mostrar el resultado de manera eficiente y con entradas grandes.

Alternativa 2:

En esta alternativa podemos encontrar que el usuario ingresara los datos o elementos por medio de un cuadro de texto, esto le facilitara al usuario en lo relacionado con el tiempo de ingreso, pero tendrá una desventaja la cual es que cuando el usuario deseara ingresar entradas grandes el cuadro de texto no será una buena funcionalidad ya que no le permitirá ver muy bien los elementos ingresados. Por otra parte, cuando hablamos de los métodos que vamos a utilizar, tanto el mergesort, quicksort y heapsort tienen una complejidad temporal muy baja en comparación con otros métodos de ordenamiento, pero una o varias desventajas que podríamos encontrar con este, es a la hora de la complejidad espacial, la cual se basa en la cantidad de memoria que utilizara el programa implementando estos algoritmos. Otra desventaja que podríamos encontrar en los algoritmos de ordenamiento es que a la hora de la implementación puede volverse algo tedioso.

Una de las ventajas de esta alternativa a la hora de mostrar el resultado obtenido es que esta nos mostrará por medio de un cuadro de texto se mostrará el arreglo ordenado y en una lista. Además, mostrará el tiempo de ejecución, el algoritmo utilizado, el número de entradas ingresadas, y la cantidad de memoria utilizada cada una en una etiqueta, esto significa que el usuario tendrá toda la información de lo que está haciendo el programa. Una desventaja sería que a la hora de mostrar entradas grandes no

Alternativa 3:

En esta alternativa podemos encontrar que el usuario ingresara los datos por medio de un campo emergente. Una de las ventajas que podríamos encontrar a la hora de implementar esta solución es que el usuario podrá ingresar los datos de una manera concisa y sin gasto de tiempo. También, por medio del campo de texto se le hará saber al usuario como ingresar los datos de una manera correcta reduciendo así el nivel de error. Una vez los elementos sean ingresados estos harán el llamado a los métodos MergeSort, o Quick Sort. Estos métodos tienen una ventaja ya que estos son muy eficientes en cuestión de tiempo y retornan una respuesta eficaz, pero a la hora de la implementación se vuelve un poco complicado de tratar. Por otra parte, también tenemos la opción de crear un árbol binario con los elementos ingresados por el usuario, la ventaja de esta opción es que ya nos estaría organizando los elementos en la lista, y esto nos facilitaría a la hora de buscar los elementos en ella. Una ventaja de utilizar un árbol es la facilidad de implementación por el medio recursivo.

Finalmente, los datos ordenados serán exportados como un archivo plano. Una de las ventajas de exportar los elementos como un archivo plano es que el usuario tendrá facilidad de consultar los datos ingresados, el método que lo ordeno, el tiempo que se demoró, la cantidad de entradas que se ingresaron y el total de memoria utilizada, cuantas veces él quiera, ya que estos estarán guardados en el equipo. Una desventaja de esto es que podría tomar un poco de tiempo dependiendo si la entrada fue grande o no.

Alternativa 4:

Pros:

- **Evita que el usuario ingrese datos de forma manual:**
Ya que el usuario ingresará los datos por medio de un archivo plano, se beneficiará de la función de generación de datos aleatorios que ofrecerá el programa. Al mismo tiempo, el software no tendrá que esperar entradas inesperadas que el usuario quiera ingresar.
- **Permite usar métodos de ordenamiento muy eficientes:**
Tanto el MergeSort, el Shellort y el HeapSort brindarán tres estrategias que ofrecen un nivel de respuesta muy alto en función del tiempo.
- **Facilita mostrar datos pequeños:**
Al implementar una JList, los datos pequeños serán muy fáciles de visualizar. El usuario podrá arrastrar el scroll y ver la lista de datos ordenados.

Contras:

- **Implementa algoritmos complicados:**
ShellSort, HeapSort son algoritmos complicados de entender y, posiblemente, de implementar.
- **Vista para entradas grandes difíciles de mostrar:**
La JList puede que no soporte entradas grandes de datos y colapse. En caso de que soporte la entrada, el usuario no tendrá claridad en la lista que está viendo pues será una lista vertical inmensa.
- **Complejidad en la sincronización de la función “generar números”:**

Como la única entrada será dada por un archivo de texto, la función que genera los números debe proveer un archivo que cumpla con los requisitos del proceso de entrada. Esto quiere decir que, el archivo creado por la función “generar números” siempre debe cumplir las especificaciones de aceptación del proceso de ingreso de datos.

Alternativa 5:

Pros:

- **Evita que el usuario ingrese datos de forma manual:**
Ya que se leerán por medio de un archivo plano los números que el usuario desea ordenar. El mismo puede usar un generador de números aleatorios o bien el ya incluido en el programa para crear dicho archivo y evitar escribirlos manualmente.
- **Múltiples estrategias para procesar el ordenamiento:**
SelectionSort, ShellSort y una tabla hash brindan un respaldo para los casos en que cierta estrategia resulte ineficiente. Por ejemplo, SelectionSort, brinda una alternativa para entradas pequeñas.
- **Visualización mejor estructurada:**
Al usar el campo de texto para mostrar la información relevante como el tiempo de ejecución, entre otras, y la lista mostrando el arreglo ordenado, el usuario podrá tener un panorama mejor estructurado del resultado.

Contras:

- **Estructura de datos nueva:**
La tabla hash da muchas incertidumbres, tanto en su implementación como en su ejecución. Ya que es un tema nuevo, es posible que nos tome mucho tiempo implementarla. Al mismo tiempo, es posible que no de los resultados que necesitemos o haya sido programada mal.
- **Sobre carga de la lista que mostrará los datos:**
Cómo se mostrará en una JList, en caso de que el usuario escoja un millón de datos, es posible que el componente no lo soporte. O bien el usuario deba usar su rueda del ratón demasiado para poder ver su lista ordenada.
- **Complejidad en la sincronización de la función “generar números”:**
Ya que el programa solo recibirá un archivo, el único método para generarlo rápidamente es el generador de datos. Este debe asegurarse de que el archivo exportado no genere ningún error al momento de ser ingresado de regreso en la entrada.

Alternativa 6:

Pros:

- **Exportación de resultados:**
El usuario tendrá la posibilidad de guardar un registro con los datos de cada prueba pues, el programa, una vez terminado el ordenamiento, exportará un reporte con los datos que se registraron durante el proceso.

- **Métodos eficientes de procesar la información:**

Al implementar un árbol binario de búsqueda, una tabla hash y el ordenamiento HeapSort, tendremos tres estrategias interesantes que podrían procesar el ordenamiento de manera eficiente y tal vez complementaria. Pues una podría ayudar a la otra.

- **Visualizaciones estructuradas de los datos:**

Mostrando los datos relevantes como el tiempo de ejecución, el algoritmo que lo soluciono y los recursos utilizados por medio de etiquetas, serán datos que son fácilmente visibles.

Contras:

- **Estructura de datos nueva:**

La tabla hash da muchas incertidumbres, tanto en su implementación como en su ejecución. Ya que es un tema nuevo, es posible que nos tome mucho tiempo implementarla. Al mismo tiempo, es posible que no de los resultados que necesitemos o haya sido programada mal.

- **Ordenamiento complejo:**

Ordenar por el método de HeapSort representa un desafío pues es algo complicado de entender e implementar.

- **Incetidumbre en la visualización de los datos:**

Es posible que el campo de texto no muestre los datos o que colapse

Alternativa 7:

En esta alternativa encontramos que el usuario tendrá la opción de ingresar los datos por un archivo plano. Una de las ventajas de ingresar los elementos por un archivo plano es que este ya viene con unos criterios de lectura y el usuario no tendrá que escribirlos en un cuadro de texto, esto reduciría el margen de error a la hora de la lectura de archivos. Por otra parte, en el mundo encontraremos que los métodos, QuickSort, MergeSort y PigeonHol Sort brindan alternativas distintas que satisfacen las deficiencias de cada uno , complementándose entre sí. Una d elas ventajas que podríamos encontrar a la hora de usar el método QuickSort, es que Requiere de pocos recursos en comparación a otros métodos de ordenamiento, en la mayoría de los casos, se requiere aproximadamente $N \log N$ operaciones y este tiene un ciclo interno extremadamente corto.Una de las desventajas que podríamos encontrar al implementar este método es que se complica la implementación si la recursión no es posible, en el peor caso, se requiere N^2 , un simple error en la implementación puede pasar sin detección, lo que provocaría un rendimiento pésimo y no es útil para aplicaciones de entrada dinámica, donde se requiere reordenar una lista de elementos con nuevos valores. Por otra parte, tenemos el método MergeSort, una de sus varias ventajas es que es estable mientras la función de mezcla sea implementada correctamente, también es muy estable cuando la cantidad de registros a acomodar es de índice bajo, en caso contrario gasta el doble del espacio que ocupan inicialmente los datos. Hablando por otro lado de sus desventajas podríamos encontrar que este está definido recursivamente. Si se deseara implementarla no recursivamente se tendría que emplear una pila y se requeriría un espacio adicional de memoria para almacenarla. Por último, tenemos el método

Fase 6. En este paso procedemos a definir los criterios que nos permitirán evaluar las alternativas de las soluciones anteriormente propuestas y con base en este resultado elegir la solución que mejor satisface las necesidades del problema planteado. Los criterios que escogimos en este caso son los que enumeramos a continuación. Al lado de cada uno se ha establecido un valor numérico con el objetivo de establecer un peso que indique cuáles de los valores posibles de cada criterio tienen más

Criterio 1: En el criterio 1 evaluaremos la precisión de la solución que se va a escoger, definiéndola en una escala numérica.

- A) La solución es Exacta—Valor: 2
- B) La solución es Aproximada –Valor: 1

Criterio 2: En el criterio 2 evaluaremos la facilidad que tendrá el usuario al momento de ingresar los datos.

- A) Fácil ----) Valor: 3
- B) Media ---) Valor: 2
- C) Algo Complicada ---) Valor:1

Criterio 4: En el criterio 4 evaluaremos el tiempo utilizado por cada algoritmo.

- A) Constante --) Valor: 1
- B) Logarítmica –) Valor: 6
- C) Lineal --) Valor: 5
- D) Cuadrática --) Valor: 4
- E) Cubica --) Valor: 3
- F) Exponencial --) Valor: 2

Criterio 5: En el criterio 5 evaluaremos el espacio en memoria utilizado por cada algoritmo.

- A) Constante --) Valor: 1
- B) Logarítmica –) Valor: 6
- C) Lineal --) Valor: 5
- D) Cuadrática --) Valor: 4
- E) Cubica --) Valor: 3
- F) Exponencial --) Valor: 2

Criterio 6: En el criterio 6 evaluaremos la facilidad con la que se muestran los datos en el programa.

- A) Fácil ----) Valor: 3
- B) Media ---) Valor: 2
- C) Algo Complicada ---) Valor:1

Alternativas	Criterio 1	Criterio 2	Criterio 3	Criterio 4	Criterio 5	Criterio 6
	Precisión de la solución	Facilidad de ingreso de datos	Facilidad de implementación del mundo	Tiempo Utilizado por cada Algoritmo	Espacio de Memoria Utilizado por cada Algoritmo	Facilidad de implementación del algoritmo
Alternativa 1		<p>Cuadro Emergente : 3</p> <p>Archivos de texto Plano: 2</p>	Medio: 2	<p>Merge Sort: $O(n \log n)$: 6</p> <p>Seleccction Sort: $O(n^2)$: 4</p> <p>Shell Sort: $O(n^{1.25})$: 2</p>	<p>Merge Sort: $O(n)$: 5</p> <p>Seleccction Sort: $O(1)$: 1</p> <p>Shell Sort: $O(1)$: 1</p>	<p>Merge Sort: 2</p> <p>Seleccction Sort: 3</p> <p>Shell Sort: 2</p>
Alternativa 2		Campo de texto: 3	Medio 2	<p>Merge Sort: $O(n \log n)$: 6</p> <p>Quizksort: $O(n^2)$: 4</p> <p>Heapsort: $O(n \log n)$: 6</p>	<p>Merge Sort: $O(n)$: 5</p> <p>Quizksort: $O(\log n)$: 6</p> <p>Heapsort: $O(1)$: 1</p>	<p>Merge Sort: 2</p> <p>Quizksort: 3</p> <p>Heapsort: 2</p>
Alternativa 3		Campo Emergente : 3	Medio : 2	<p>Merge Sort: $O(n \log n)$: 6</p> <p>Quizksort: $O(n^2)$: 4</p> <p>Arbol Binario: $O(\log n)$: 6</p>	<p>Merge Sort: $O(n)$: 5</p> <p>Quizksort: $O(\log n)$: 6</p>	<p>Merge Sort: 2</p> <p>Quizksort: 3</p> <p>ArbolBinario: 3</p>
Alternativa 4		Archivo de texto plano: 2	Medio: 2	<p>Merge Sort: $O(n \log n)$: 6</p> <p>Shell Sort: $O(n^{1.25})$: 2</p> <p>Heapsort: $O(n \log n)$: 6</p>	<p>Merge Sort: $O(n)$: 5</p> <p>Shell Sort: $O(1)$: 1</p> <p>Heapsort: $O(1)$: 1</p>	<p>Merge Sort: 2</p> <p>Shell Sort: 2</p> <p>Heapsort: 2</p>

Alternativa 5		Archivo de texto Plano: 2	Medio: 2	Seleccction Sort: $O(n^2)$: 4 Shell Sort: $O(n^{1.25})$: 2 Tablas Hash: $O(1)$: 1	Seleccction Sort: $O(1)$: 1 Shell Sort: $O(1)$: 1 Tablas Hash: $O(c)$: 1	Seleccction Sort: 3 Shell Sort: 2 Tablas Hash: 1
Alternativa 6		Campo de texto: 3	Medio: 2	Arbol Binario: $O(\log n)$: 6 Tablas Hash: $O(1)$: 1 Heapsort: $O(n \log n)$: 6	Tablas Hash: $O(c)$: 1 Heapsort: $O(1)$: 1	ArbolBinario: 3 Tablas Hash: 1 Heapsort: 2
Alternativa 7		Archivo de texto plano: 2 Campo de texto: 3	Medio: 2	Quizksort: $O(n^2)$: 4 Merge Sort: $O(n \log n)$: 6	Quizksort: $O(\log n)$: 6 Merge Sort: $O(n)$: 5	Quizksort: 3 Merge Sort: 2

Fase 7. Evaluación y selección de la mejor solución

Alternativa3:

Categoría2(3+2)+Categoría3(2)+Categoría4(6+4+2)+Categoría5(5+1+1)+Categoría6(2+3+2)+=37

Alternativa 4:

Categoría 2(2)+Categoría3(2)+Categoría 4(6+2+6)+ Categoría5(5+1+1)+Categoría 6(2+2+2)+=30

Alternativa 7:

Categoría 2(2+3)+Categoría3(2)+Categoría 4(4+6+6)+ Categoría5(6+5+1)+Categoría 6(3+2+2)+=42

En esta fase lo que procedimos a hacer fue evaluar la mejor solución, teniendo en cuenta la tabla de criterios anteriormente definida y descartar las ideas que no alcanzaran el puntaje promedio para pasar. Al final de la evaluación encontramos que las mejores soluciones son la Alternativa 3, 4 y 7, descartando así las Alternativas 1, 2, 5 y 6. Con base en el puntaje obtenido lograremos escoger la mejor solución entre las tres restantes y así poder implementar con todos sus requerimientos correspondientes.

Fase 8. Diseño preliminar de cada idea no descartada: (Modelos de Simulación)

Alternativa 3:

Miguel y Jhonatan

Enter elements

Generate

Element to be sorted...

Please enter the elements with a "-" between each other

1-4-7-9-2-15-12-3-78

Accept

What interval do you want?

Lower limit

...

Upper limit

...

Accept

What type of data?

☒ Integers ☐ Floats

Accept

How would you like the elements to be?

☐ Sorted ☒ Conversely ordered ☐ not sorted in a percentage

Do you want the numbers to be repeated?

☒ Yes ☐ No

Accept

What percentage?

No sorted in a percentage

95%

Accept

Element to be sorted...

Please enter the elements with a "-" between each other

1-4-7-9-2-15-12-3-78

Accept

En esta alternativa el usuario tendrá dos opciones, la primera será donde el ingresará los elementos manualmente, en este caso (Enter Elements). Una vez el usuario de click allí, el programa le mostrara una opción en donde el podrá organizarlos, también le explicara como deberán de ser organizados, para que no haya margen de error a la hora de ingreso de elementos.

What type of data?

☒ Integers ☐ Floats

Accept

Por otra parte, cuando el usuario haga click en el botón generar, este le mostrara otra opción en donde el tendrá que escoger que tipo de datos le gustaría ordenar, ya sea Integers o Floats.

Do you want the numbers to be repeated?

☒ Yes ☐ No

Accept

Una vez el usuario decida que tipo de elementos le gustaría generar, aparecerá otra opción por parte del sistema en donde le preguntara al usuario que si le gustaría que los elementos se repitieran.

What interval do you want?

Lower limit

Upper limit

Accept

Cuando el usuario de la opción de ☒ Yes, el programa le preguntara cual es el intervalo en que los números a ordenar debería de estar, desde donde debe de empezar (Lower Limit) hasta a donde debería llegar (Upper Limit).

How would you like the elements to be?

☐ Sorted

☒ Conversely ordered

☐ not sorted in a percentage

Una vez el usuario haya ingresado tanto el limite inferior como el limite superior, el programa le mostrara una opción en la cual le preguntara al usuario como le gustaría los elementos, ya se (Ordenados, Inversamente Ordenados o un Porcentaje Ordenado)

What percentage?

No sorted in a percentage

95%

Accept

Si el usuario llegara a escoger la opción de (Valores desordenados en un porcentaje) el programa le mostrara otra opción en donde el usuario tendrá que elegir qué porcentaje de los elementos le gustaría que estuvieran desordenados.



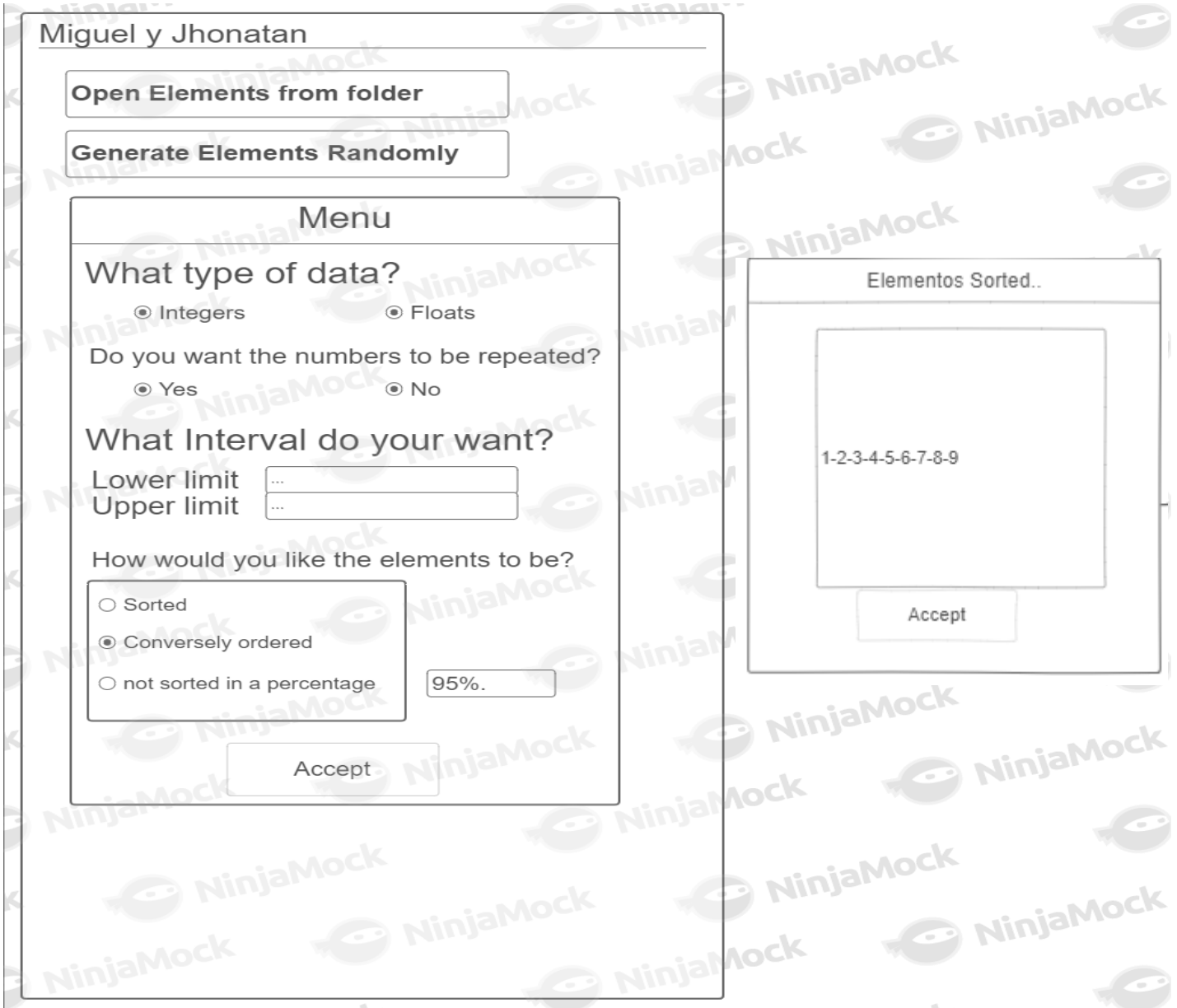
Elementos Sorted..

1-2-3-4-5-6-7-8-9

Accept

Una vez el usuario haya respondido cada pregunta del programa, este le arrojará por medio de un archivo plano y se mostrará al usuario, los elementos ordenados, separados por un “-”.

Alternativa 4:



Miguel y Jhonatan

Open Elements from folder

Generate Elements Randomly

Menu

What type of data?

☐ Integers ☒ Floats

Do you want the numbers to be repeated?

☒ Yes ☐ No

What Interval do you want?

Lower limit

Upper limit

How would you like the elements to be?

☐ Sorted

☒ Conversely ordered

☐ not sorted in a percentage

Accept

Elementos Sorted..

1-2-3-4-5-6-7-8-9

Accept

En esta alternativa el usuario tendrá dos opciones, una en la cual el tendrá que ingresar los elementos por medio de un archivo plano. Una vez ingresados los elementos, este los ordenará y serán mostrados por medio de una lista. Por otra parte, el Usuario también contará con una opción en la cual el podrá generar los elementos aleatoriamente, si el usuario escoge esta opción el programa le mostrará un menú, en donde el tendrá que escoger el tipo de dato, si le gustaría que los elementos se repitieran, de donde hasta donde le gustaría que los elementos estuvieran y una última opción en donde el programa le preguntará al usuario como le gustaría que le generara los elementos, ya sea

ordenados, inversamente ordenados, y un porcentaje de desordenamiento. Para concluir, los elementos serán mostrados por medio de una lista.

Alternativa 7:

