

CARINE CASA GRANDE

**COMPARAÇÃO ENTRE O DESEMPENHO DE APLICAÇÕES PARA
SMARTPHONES DESENVOLVIDAS EM FLUTTER E REACT NATIVE:
UMA ANÁLISE UTILIZANDO ALGORITMOS DE ORDENAÇÃO**

CARINE CASA GRANDE

**COMPARAÇÃO ENTRE O DESEMPENHO DE APLICAÇÕES PARA
SMARTPHONES DESENVOLVIDAS EM FLUTTER E REACT NATIVE:
UMA ANÁLISE UTILIZANDO ALGORITMOS DE ORDENAÇÃO**

Trabalho de Conclusão de Curso apresentado ao Curso Ciência da Computação do Centro Universitário Filadélfia – UniFil, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação.

Orientadora: Prof.^a Me. Simone Sawasaki Tanaka

Londrina
2021

CARINE CASA GRANDE

**COMPARAÇÃO ENTRE O DESEMPENHO DE APLICAÇÕES PARA
SMARTPHONES DESENVOLVIDAS EM FLUTTER E REACT NATIVE:
UMA ANÁLISE UTILIZANDO ALGORITMOS DE ORDENAÇÃO**

Trabalho de Conclusão de Curso apresentado
ao Curso de Ciência da Computação do Centro
Universitário Filadélfia – UniFil, como requisito
parcial à obtenção do título de Bacharel em
Ciência da Computação.

Banca Examinadora

Orientadora: Prof.^a Me. Simone Sawasaki
Tanaka
Centro Universitário Filadélfia – UniFil

Prof. Componente de Banca
Centro Universitário Filadélfia – UniFil

Prof. Componente de Banca
Centro Universitário Filadélfia – UniFil

Londrina, ____ de _____ de ____.

GRANDE, Carine Casa. **Comparação entre o desempenho de aplicações para *smartphones* desenvolvidos em Flutter e React Native**: Uma análise utilizando algoritmos de ordenação. 2021. 59 f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação). Centro Universitário Filadélfia – UniFil, Londrina, 2021.

RESUMO

O uso de *smartphones* tem crescido consideravelmente nos últimos anos, tornando o mercado de desenvolvimento de aplicativos cada vez mais forte. Atualmente, existem dois principais sistemas operacionais para esses dispositivos, o iOS e o Android. Entretanto, como os dois possuem linguagens nativas diferentes, sendo o Swift e o Objective-C para o primeiro e Java e Kotlin para o segundo, o desenvolvimento nativo se mostra de alto custo. Alguns *frameworks* podem ser utilizados para desenvolver de forma híbrida um aplicativo, isto é, um único código que gera aplicativos para mais de um sistema operacional, diminuindo este custo e tornando o desenvolvimento mais prático. Este trabalho analisa e compara o desempenho dos aplicativos híbridos criados em Flutter e em React Native, dois dos principais *frameworks* que possuem tal objetivo. Para isso, foi criado um aplicativo para executar algoritmos de ordenação, onde o tempo de execução foi medido e os resultados comparados para cada sistema operacional. Embora trabalhos correlatos tenham determinado o Flutter como o *framework* que entrega o aplicativo com maior desempenho, os resultados mostram que o React Native se mostrou mais ágil nos testes realizados com os algoritmos de ordenação.

Palavras-chave: Flutter. React Native. Desenvolvimento híbrido. *Framework*.

GRANDE, Carine Casa. **Comparison between the performance of smartphone applications developed in Flutter and React Native:** An analysis using sorting algorithms. 2021. 59 f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação). Centro Universitário Filadélfia – UniFil, Londrina, 2021.

ABSTRACT

The use of smartphones has grown considerably in recent years, making the application development market stronger and stronger. There are currently two main operating systems for these devices, iOS and Android. However, as the two have different native languages, being Swift and Objective-C for the first and Java and Kotlin for the second, native development proves to be costly. Some frameworks can be used to hybrid develop an application, that is, a single code that generates applications for more than one operating system, reducing this cost and making development more practical. This work analyzes and compares the performance of hybrid applications created in Flutter and React Native, two of the main frameworks that have this objective. For this, an application was created to run sorting algorithms, where the execution time was measured and the results compared for each operating system. Although related works have determined Flutter as the framework that delivers the application with the highest performance, the results show that React Native proved to be more agile in tests performed with the sorting algorithms.

Keywords: Flutter. React Native. Hybrid development. Framework.

LISTA DE ILUSTRAÇÕES

Figura 1 – Algoritmos de ordenação mais pesquisados nos últimos cinco anos	15
Figura 2 – Pesquisa por outros algoritmos de ordenação nos últimos cinco anos ..	16
Figura 3 – Ilustração do funcionamento do algoritmo Bubble Sort.....	17
Figura 4 – Implementação do algoritmo Bubble Sort regular em JavaScript	18
Figura 5 – Implementação do algoritmo Bubble Sort otimizado em JavaScript.....	18
Figura 6 – Ilustração do funcionamento do algoritmo Quick Sort.....	19
Figura 7 – Implementação do algoritmo Quick Sort em JavaScript	19
Figura 8 – Ilustração do funcionamento do algoritmo Merge Sort.....	20
Figura 9 – Implementação do algoritmo Merge Sort em JavaScript	21
Figura 10 – Ilustração do funcionamento do algoritmo Insertion Sort.....	21
Figura 11 – Implementação do algoritmo Insertion Sort em JavaScript.....	22
Figura 12 – Ilustração do funcionamento do algoritmo Selection Sort	22
Figura 13 – Implementação do algoritmo Selection Sort em JavaScript.....	23
Figura 14 – Busca por React Native, Flutter, Ionic e Xamarin	25
Figura 15 – Logo do React Native.....	26
Figura 16 – Elemento em React Native no Android (direita) e no iOS (esquerda)...	26
Figura 17 – Logo do Flutter	27
Figura 18 – Elemento em Flutter no Android (direita) e no iOS (esquerda).....	27
Figura 19 – Teste de execução do algoritmo Gauss-Legendre no Android e iOS ...	28
Figura 20 – Teste de execução do algoritmo Borwein no Android e iOS.....	28
Figura 21 - Opções para cada parâmetro utilizado nos testes.....	31
Figura 22 – Protótipo do aplicativo	34
Figura 23 – Resultado do aplicativo construído em Flutter no iOS	35
Figura 24 – Resultado do aplicativo construído em Flutter no Android.....	35
Figura 25 – Resultado do aplicativo construído em React Native no iOS.....	36
Figura 26 – Resultado do aplicativo construído em React Native no Android.	36
Figura 27 – Média de tempo utilizando a lista pré-determinada por algoritmo de ordenação	38
Figura 28 – Média de tempo utilizando a lista aleatória por algoritmo de ordenação	39

Figura 29 – Média de tempo utilizando a lista pré-determinada por algoritmo de ordenação	40
Figura 30 – Média de tempo utilizando a lista aleatória por algoritmo de ordenação	41

LISTA DE TABELAS

Tabela 1 – Flutter vs. React Native de acordo com De Almeida (2019).....	29
Tabela 2 – Especificação dos dispositivos utilizados para realizar os testes	32
Tabela 3 – Média de tempo utilizando a lista pré-determinada por quantidade de elementos.....	38
Tabela 4 – Média de tempo utilizando a lista aleatória por quantidade de elementos	39
Tabela 5 – Média de tempo utilizando a lista pré-determinada por quantidade de elementos.....	40
Tabela 6 – Média de tempo utilizando a lista aleatória por quantidade de elementos	41

LISTA DE SIGLAS E ABREVIATURAS

APP	Aplicativo
CPU	Unidade Central de Processamento
CSS	Cascading Style Sheets
HTML	HyperText Markup Language
IBGE	Instituto Brasileiro de Geografia e Estatística
JS	JavaScript
PDA	Assistente Pessoal Digital
SO	Sistema Operacional
UI	Interface do Usuário
UNIFIL	Centro Universitário Filadélfia

SUMÁRIO

1	INTRODUÇÃO	11
1.1	Objetivos	11
1.1.1	Objetivo Geral	11
1.1.2	Objetivos Específicos	12
2	FUNDAMENTAÇÃO TEÓRICA.....	13
2.1	Dispositivos Móveis	13
2.2	Sistemas Operacionais	13
2.3	Linguagem de Programação	14
2.4	Algoritmos	15
2.4.1	Complexidade de algoritmos	16
2.4.2	Bubble Sort	17
2.4.3	Quick Sort.....	19
2.4.4	Merge Sort	20
2.4.5	Insertion Sort	21
2.4.6	Selection Sort.....	22
2.5	Desenvolvimento Móvel	23
2.6	Frameworks	25
2.6.1	React Native.....	25
2.6.2	Flutter	26
2.7	Flutter vs. React Native	27
3	METODOLOGIA	31
4	DESENVOLVIMENTO DO APLICATIVO.....	33
4.1	Ambiente de Desenvolvimento	33
4.2	Protótipo do Aplicativo	33
4.3	Desenvolvimento em Flutter	34
4.4	Desenvolvimento em React Native.....	36
5	RESULTADO E DISCUSSÃO	38

5.1	Android	38
5.2	iOS	40
5.2.3	Análise	42
6	CONCLUSÃO	43
	REFERÊNCIAS	44
	APÊNDICES	51
	Apêndice A – Teste do Flutter realizado com lista pré-determinada no iOS.....	52
	Apêndice B – Teste do Flutter realizado com lista pré-determinada no Android.	53
	Apêndice C – Teste do React Native realizado com lista pré-determinada no iOS.....	54
	Apêndice D – Teste do React Native realizado com lista pré-determinada no Android.	55
	Apêndice E – Teste do Flutter realizado com lista aleatória no iOS.....	56
	Apêndice F – Teste do Flutter realizado com lista aleatória no Android.	57
	Apêndice G – Teste do React Native realizado com lista aleatória no iOS.....	58
	Apêndice H – Teste do React Native realizado com lista pré-determinada no Android.	59

1 INTRODUÇÃO

O uso dos aparelhos móveis tem crescido cada vez mais, um exemplo disso é o resultado das pesquisas do Instituto Brasileiro de Geografia e Estatística (IBGE) de 2019, que mostraram que 81% das pessoas com mais de 10 anos possuíam um telefone celular ou um *smartphones* para uso pessoal. Com base nisso, vemos que o desenvolvimento de aplicações móveis é uma área que tem crescido consideravelmente no mercado de programação por conta da demanda que ela possui.

Atualmente, existem dois principais sistemas operacionais (SO) utilizados pelos usuários de *smartphones*, sendo eles o Android e iOS. Juntos, estes sistemas operacionais lideraram o mercado mundial em 2020, somando mais de 98% na participação (STATCOUNTER, 2021).

Logo, para atingir a maior parte do mercado é preciso que os aplicativos sejam desenvolvidos pensando em ambas as plataformas. Entretanto, elas diferem entre si, o que torna o desenvolvimento nativo caro e mais demorado. De modo a agilizar o desenvolvimento e facilitar a manutenção dos aplicativos foram criadas categorias de desenvolvimentos que permitem o desenvolvimento híbrido, isto é, tecnologias que permitem que um único código sirva diversos sistemas.

Dentre essas tecnologias, podemos citar o Ionic, o Xamarin, o React Native e o Flutter como exemplo, sendo que as duas últimas serão abordadas no trabalho por serem as mais discutidas nos últimos tempos. Foi comparado o desempenho de aplicativos desenvolvidos com o Flutter e com o React Native tanto no iOS quanto no Android a fim de determinar o *framework* que entrega o aplicativo de maior desempenho.

1.1 Objetivos

1.1.1 Objetivo Geral

Escolher corretamente qual tecnologia será utilizada para desenvolver um sistema é essencial para que ele forneça uma melhor experiência para o usuário e um bom desempenho. Este trabalho visa comparar o Flutter e o React Native com base no desempenho dos mesmos durante a execução de algoritmos de ordenação de

modo a determinar a melhor alternativa de *framework* para ser utilizada no desenvolvimento híbrido.

1.1.2 Objetivos Específicos

- Definir os termos que serão utilizados ao decorrer do trabalho.
- Explicar sobre os algoritmos de ordenação escolhidos para testar o desempenho dos *frameworks*.
- Exemplificar as principais características de cada um dos *frameworks* estudados.
- Desenvolver um aplicativo onde será possível executar os algoritmos de ordenação escolhidos.
- Medir o tempo de execução dos algoritmos.
- Comparar os resultados obtidos em cada sistema operacional e para cada algoritmo executado.
- Determinar o *framework* que obteve os melhores resultados para cada sistema operacional.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Dispositivos Móveis

De acordo com Figueiredo e Nakamura (2003), os dispositivos móveis possuem a capacidade de realizar processamento de dados e trocar informações via rede, além de ser facilmente transportado pelo usuário sem a necessidade de utilizar cabos de rede ou energia.

Até pouco tempo esse termo estava associado somente aos celulares, notebooks e as assistentes pessoais digitais (PDA), porém atualmente outros aparelhos também se encaixam na categoria como os *smartwatches*, leitores de livros digitais, *smartphones*, dentre outros.

Em tradução literal, *smartphone* significa telefone inteligente e é uma evolução dos aparelhos celulares. Eles permitem acesso a aplicativos (*apps*) de terceiros, acesso à internet e capacidade de armazenar dados em seus sistemas operacionais (FEIJÓ; GONÇALVES; GOMEZ, 2013).

2.2 Sistemas Operacionais

Os sistemas operacionais (SO) são *softwares* complexos que gerenciam os componentes de um dispositivo, além de fornecer uma interface gráfica que facilita a interação do usuário com ele (MENDONÇA; BITTAR; DIAS, 2011). Atualmente os principais sistemas disponíveis para *smartphones* são o Android e iOS.

De acordo com Pereira (2016), o Android é um sistema operacional criado em 2003 por Andy Rubin, Rich Miner, Nick Sears e Chris White, empresários do ramo de tecnologia que fundaram a Android Inc., onde objetivo era desenvolver um sistema para câmeras digitais, mas após um tempo focou em SO para *smartphones*.

Em 2005, a Google adquiriu diversas startups de tecnologia sendo uma delas a Android Inc., momento em que foi criado a Google Mobile Division, setor responsável por pesquisas em tecnologia móvel.

Sua primeira versão oficial foi lançada em setembro de 2008 e atualmente encontra-se na versão estável Android 11. Além de *smartphones*, está presente também em *smartwatches*, carros e televisões (PEREIRA, 2016). Por possuir código

aberto, serve de base para a criação de sistemas customizados, como é o caso do MIUI, versão desenvolvida pela marca chinesa Xiaomi.

Segundo um levantamento do StatCounter (2021), no ano de 2020 o Android teve uma participação de 72,18% no mercado mundial de sistemas operacionais móveis, e, só no Brasil, sua participação foi de 85,82%. Quando considerado todos os sistemas operacionais disponíveis no mercado mundial, independente da plataforma, sua participação foi de 41,44%.

Já o iOS foi lançado em 2007 pela Apple e desenvolvido especificamente para o iPhone. Uma de suas principais características, presente até os dias atuais, é o fato de não permitir que o sistema seja executado em *hardwares* de terceiros, somente em produtos da própria marca (SARTORELI; UNO, 2013).

Sua primeira versão se chamava iPhone OS, porém a partir da quarta versão, lançada em 2010, passou a se chamar iOS (SALUTES, 2019). Atualmente, a versão estável mais recente é o iOS 15 e está presente tanto nos iPhones quanto nos iPods.

Em 2020, conforme dados da StatCounter (2021), o sistema participou em 26,96% do mercado mundial de sistemas para dispositivos móveis e em 13,87% no mercado brasileiro, já considerando todas as plataformas, participou mundialmente em 16,53%.

2.3 Linguagem de Programação

As linguagens de programação são métodos padronizados utilizados para implementar um código-fonte. De acordo com Roveda (2019), elas possuem regras de sintaxe (escrita) e semântica (significado), além de poderem ser de alto nível ou de baixo nível. As de alto nível são aquelas mais facilmente lidas por um humano, enquanto as de baixo nível se aproximam mais à linguagem da máquina. Algumas das linguagens existentes são o JavaScript (JS) e o Dart.

O JS é uma linguagem criada em 1995 por Brendan Eich a pedido da Netscape. Segundo a Mozilla Foundation (2021), ele é uma linguagem de programação leve, interpretada e orientada a objetos. Possui tipagem dinâmica, ou seja, não é necessário definir o tipo da variável que será declarada e, com o HyperText Markup Language (HTML) e o Cascading Style Sheets (CSS), participa das principais tecnologias utilizadas no desenvolvimento *web*.

Já o Dart foi criado em 2011 pela Google e seu objetivo era substituir o JavaScript na tríade do desenvolvimento *web*. Sendo uma linguagem fortemente tipada e de alto nível, não obteve sucesso em seu objetivo inicial, entretanto tem ganhado força por conta do Flutter, um *framework* de desenvolvimento híbrido (GUEDES, 2020).

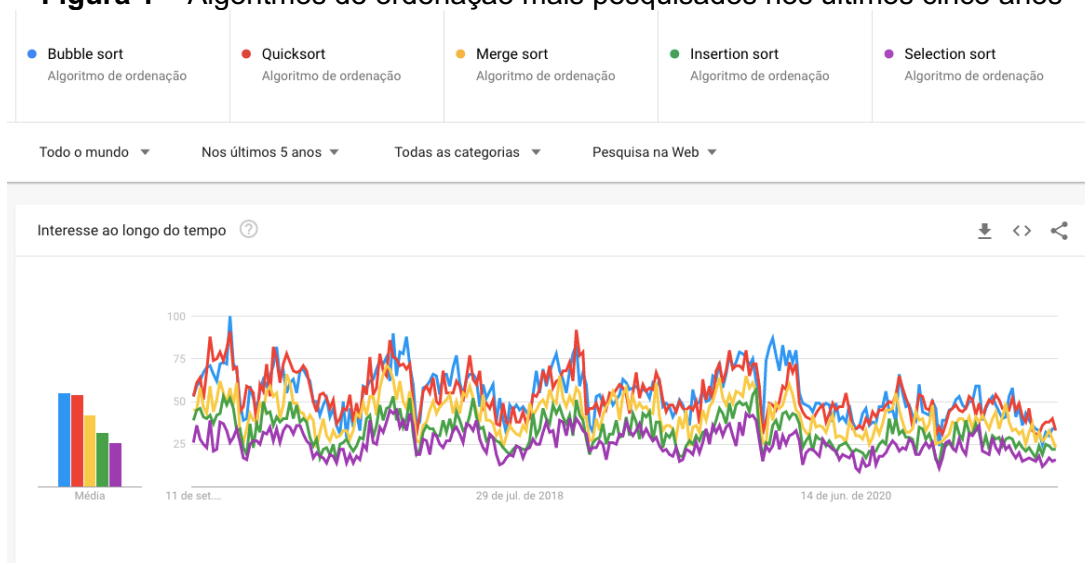
2.4 Algoritmos

De acordo com Forbellone e Eberspächer (2005), o termo algoritmo pode ser entendido como uma sequência de instruções que visam alcançar um objetivo bem definido. Para construir um algoritmo, é necessário entender o problema a ser resolvido, definir os dados de entrada, as operações que serão executadas e os dados de saída (GUEDES, 2014).

Alguns algoritmos diferentes podem ter sido criados para resolver o mesmo problema e normalmente diferem em questão de eficiência. Cormen et al. (2012) afirma que analisar um algoritmo significa prever os recursos que ele necessita, sendo, frequentemente, medido o tempo de execução.

Borin (2020) afirma que a busca pelo método de ordenação mais eficiente é um dos problemas principais no desenvolvimento de algoritmos. Os algoritmos de ordenação são aqueles que visam colocar os elementos de uma dada sequência em certa ordem (HONORATO, 2013).

Figura 1 – Algoritmos de ordenação mais pesquisados nos últimos cinco anos

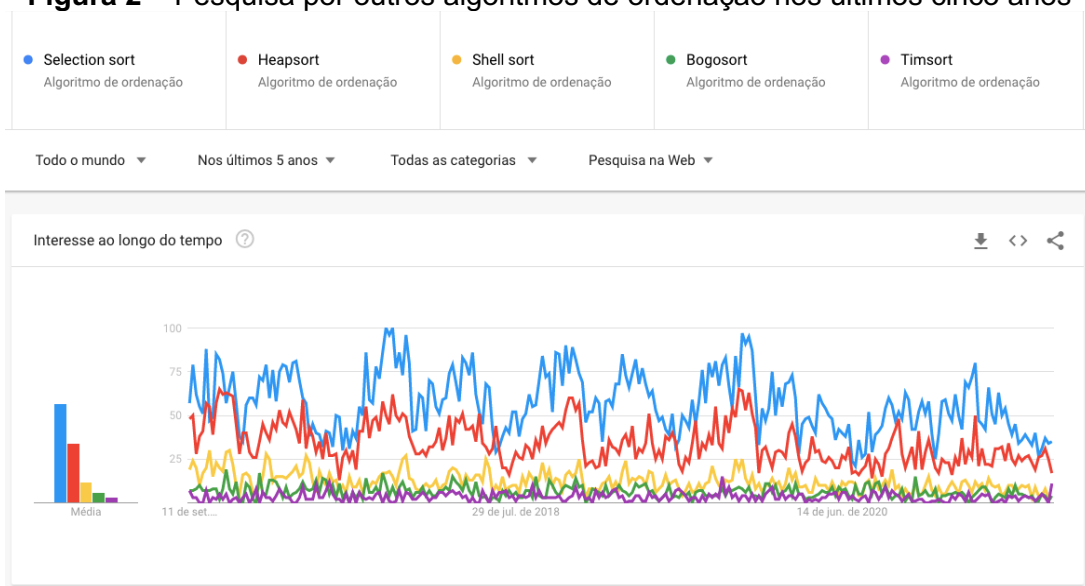


Fonte: Google Trends (2021)

Existem diversos algoritmos de ordenação, sendo que o *Bubble Sort*, *Quick Sort*, *Merge Sort*, *Insertion Sort* e *Selection Sort* são os mais buscados na ferramenta Google nos últimos cinco anos conforme o Google Trends (2021), como podemos observar na Figura 1.

Já na Figura 2, podemos ver como as buscas por outros algoritmos de ordenação, como o *Heap Sort* e o *Shell Sort*, em comparação ao *Selection Sort*, o menos pesquisado da Figura 1, são consideravelmente menores, o que demonstra como os primeiros são mais estudados e utilizados.

Figura 2 – Pesquisa por outros algoritmos de ordenação nos últimos cinco anos



Fonte: Google Trends (2021)

2.4.1 Complexidade de algoritmos

A complexidade de algoritmos estuda e define quanto eficiente é um algoritmo em relação ao número de operações necessárias para finalizar a tarefa. No caso dos algoritmos de ordenação, a ordem inicial do vetor pode fazer com que o número de operações varie (JÚNIOR, 2015).

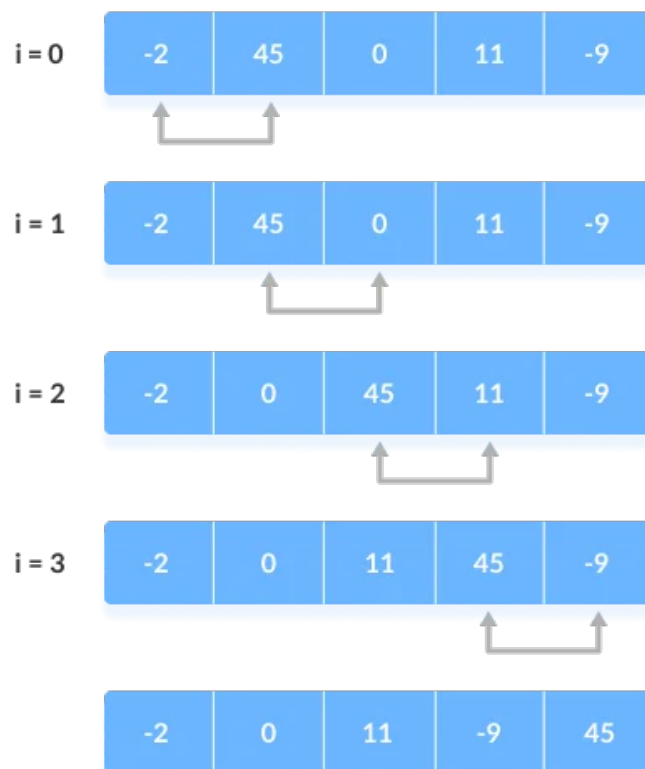
Sempre que possível, o estudo deve ser feito considerando três cenários, sendo eles o melhor caso, que corresponde ao menor número de operações realizadas, o pior caso, que corresponde ao maior número de operações realizadas e o caso médio, que corresponde à média do número de operações realizadas.

De acordo com Pereira (2019), os cientistas da computação adotaram a notação *Big – O* para descrever o comportamento de um algoritmo, utilizado para delimitar a curva de crescimento para tempo ou espaço do algoritmo.

2.4.2 *Bubble Sort*

Ascencio e Araújo (2010) explica que o algoritmo *Bubble Sort* efetua comparações entre os dados armazenados em um vetor de tamanho n . sendo que cada elemento na posição i será comparado com o elemento de posição $i + 1$. No caso de uma ordenação crescente, quando um elemento na posição i é maior que o elemento na posição $i + 1$, é realizada uma troca de posições dos elementos, como podemos visualizar na Figura 3.

Figura 3 – Ilustração do funcionamento do algoritmo *Bubble Sort*



Fonte: Programiz (2021)

Na Figura 4, observamos o algoritmo regular do *Bubble Sort*, onde o vetor será percorrido por inteiro quantas vezes for necessário, até que não haja nenhuma troca entre os elementos, o que o torna ineficiente para vetores com um tamanho muito grande.

Figura 4 – Implementação do algoritmo *Bubble Sort* regular em JavaScript

```

1  function bubbleSortRegular(vector) {
2      for (let j = 0; j < vector.length; j++) {
3          for (let i = 0; i < vector.length; i++) {
4              if (vector[i] > vector[i + 1]) {
5                  let aux = vector[i];
6                  vector[i] = vector[i + 1];
7                  vector[i + 1] = aux;
8              }
9          }
10     }
11 }

```

Fonte: Programiz (2021) (Adaptado pelo Autor, 2021)

Há também a forma otimizada deste algoritmo, como podemos ver na Figura 5, sendo que nela é utilizada uma variável que determina se naquele laço houve alguma troca. Se sim, o processo continua e caso nenhuma troca tenha sido feita, o processo é encerrado.

Figura 5 – Implementação do algoritmo *Bubble Sort* otimizado em JavaScript

```

1  function bubbleSortOptimized(vector) {
2      var hasChange = true;
3      while (hasChange) {
4          hasChange = false;
5          for (let i = 0; i < vector.length; i++) {
6              if (vector[i] > vector[i + 1]) {
7                  let aux = vector[i];
8                  vector[i] = vector[i + 1];
9                  vector[i + 1] = aux;
10                 hasChange = true;
11             }
12         }
13     }
14 }

```

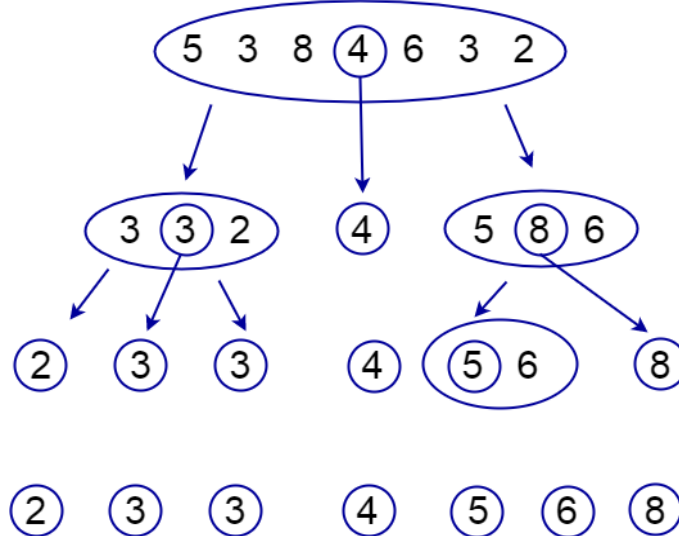
Fonte: Programiz (2021) (Adaptado pelo Autor, 2021)

De acordo com Dos Santos (2020), o *Bubble Sort* tem complexidade $O(n^2)$ tanto no caso médio quanto no pior caso, enquanto no melhor caso, onde nenhuma troca é necessária, é de $O(n)$.

2.4.3 Quick Sort

Este algoritmo consiste em dividir o arranjo inicial de elementos em subgrupos, sendo que a ordenação dos elementos se dará à medida que ocorram as divisões (DA ROSA; ANTONIAZZI, 2012).

Figura 6 – Ilustração do funcionamento do algoritmo *Quick Sort*



Fonte: Nehra (2021)

Figura 7 – Implementação do algoritmo *Quick Sort* em JavaScript

```

1  function quickSort(vector) {
2    if (vector.length <= 1) return vector;
3
4    let left = [];
5    let right = [];
6
7    let middle = Math.trunc(vector.length / 2);
8    let pivot = vector[middle];
9
10   for (let i = 0; i < vector.length; i++) {
11     if (i !== middle) {
12       if (vector[i] < pivot) left.push(vector[i]);
13       else right.push(vector[i]);
14     }
15   }
16
17   let pivotArray = [pivot];
18   return quickSort(left).concat(pivotArray, quickSort(right));
19 }

```

Fonte: Lichao (2014) (Adaptado pelo Autor, 2021)

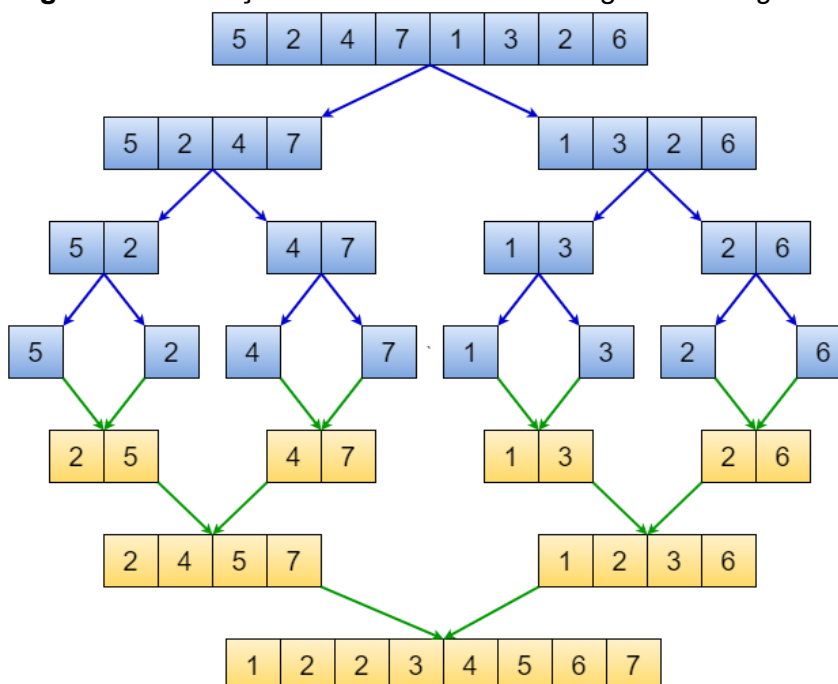
Da Fonseca et al. (2017) aponta que o vetor será ordenado recursivamente, onde ele será dividido e um elemento pivô é escolhido no meio, sendo os elementos

que forem menores que o pivô posicionados à esquerda do mesmo e os maiores à direita, como podemos visualizar na Figura 6.

De acordo com o Farias, Da Silva e Neto (2012), este algoritmo tem complexidade de tempo $O(n)$ no pior caso e $O(n \log n)$ no médio e melhor caso. Na Figura 7 vemos a implementação do *Quick Sort* em JavaScript, onde após todas as divisões os resultados são concatenados, voltando a ser somente um vetor, agora ordenado. Embora nesta a implementação o pivô escolhido seja o elemento central do vetor, isso não é uma regra, podendo ser escolhido qualquer elemento como pivô.

2.4.4 Merge Sort

Figura 8 – Ilustração do funcionamento do algoritmo *Merge Sort*



Fonte: Gupta (2020)

Segundo Souza, Ricarte e Lima (2017), este algoritmo tem como objetivo a reordenação do vetor através da divisão, intercalação e união dos elementos existentes, isto é, o algoritmo divide de forma recursiva o vetor em duas metades até que isso não seja mais possível, depois os subgrupos são comparados e combinados até que esteja todo ordenado, como podemos ver na Figura 8.

A complexidade deste algoritmo é de $O(n \log n)$ em todos os casos (PROGRAMIZ, 2021). Podemos ver na Figura 9, a implementação do *Merge Sort* em JavaScript.

Figura 9 – Implementação do algoritmo *Merge Sort* em JavaScript

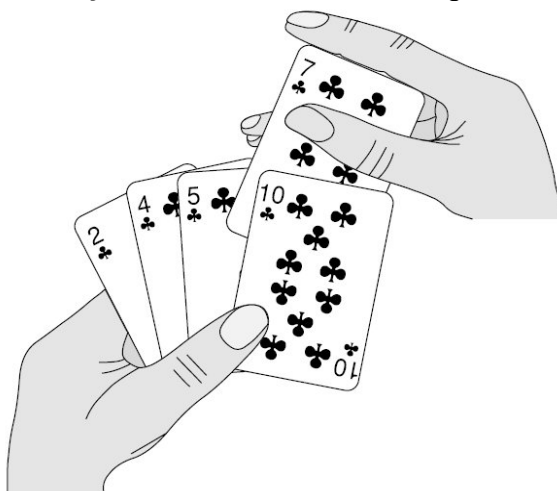
```

1  function mergeSort(vector) {
2      if (vector.length < 2) return vector;
3
4      var middle = Math.round(vector.length / 2);
5      var left = vector.slice(0, middle);
6      var right = vector.slice(middle, vector.length);
7
8      return merge(mergeSort(left), mergeSort(right));
9  }
10
11 function merge(left, right) {
12     var vector = [];
13
14     while (left.length && right.length) {
15         if (left[0] <= right[0]) vector.push(left.shift());
16         else vector.push(right.shift());
17     }
18
19     while (left.length) vector.push(left.shift());
20
21     while (right.length) vector.push(right.shift());
22
23     return vector;
24 }

```

Fonte: Veenstra (2016) (Adaptado pelo Autor, 2021)

2.4.5 *Insertion Sort*

Figura 10 – Ilustração do funcionamento do algoritmo *Insertion Sort*

Fonte: Cormen et al. (2012)

Conforme comparação feita por Cormen et al. (2012) e demonstrado na Figura 10, este algoritmo funciona da maneira que a maioria das pessoas ordenam as cartas de um baralho durante um jogo: as cartas ficam viradas na mesa, e uma a uma é

retirada e posicionada no lugar correto nas mãos e para encontrar esta posição correta, a comparação é feita com cada uma das cartas que já estão posicionadas, da direita para a esquerda.

No melhor caso a complexidade do *Insertion Sort* é de $O(n)$ enquanto no pior e no médio caso é de $O(n^2)$ (DIAS; VILELA; JÚNIOR, 2014). Na Figura 11, é possível ver a implementação deste algoritmo em JavaScript.

Figura 11 – Implementação do algoritmo *Insertion Sort* em JavaScript

```

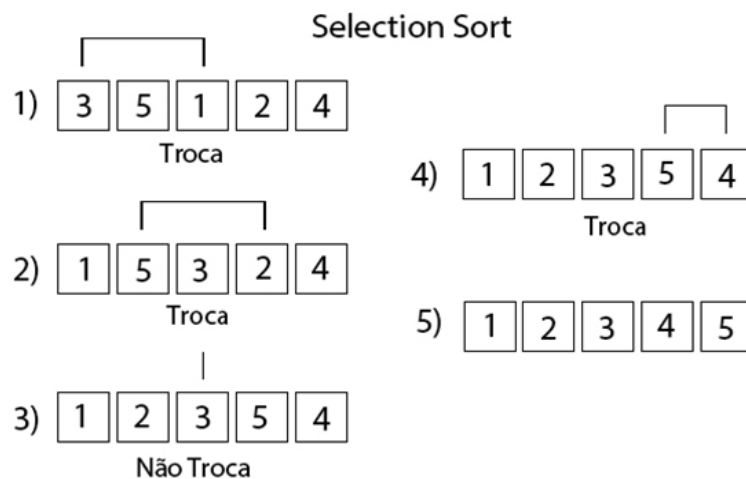
1  function insertionSort(vector) {
2    for (let i = 1; i < vector.length; i++) {
3      let key = vector[i];
4      let j = i - 1;
5
6      while (j >= 0 && vector[j] > key) {
7        vector[j + 1] = vector[j];
8        j = j - 1;
9      }
10
11     vector[j + 1] = key;
12   }
13 }

```

Fonte: GeeksforGeeks (2021) (Adaptado pelo Autor, 2021)

2.4.6 Selection Sort

Figura 12 – Ilustração do funcionamento do algoritmo *Selection Sort*



Fonte: Guimarães (2014)

Este algoritmo seleciona o menor item do vetor e o troca com o item que está na primeira posição do vetor, repetindo estas operações até que o vetor esteja totalmente ordenado (ZIVIANI, 1999), como podemos ver na Figura 12.

De acordo com Dias, Vilela e Júnior (2014), este algoritmo apresenta complexidade $O(n^2)$ no pior, melhor e médio caso. Na Figura 13, é possível ver a implementação do *Selection Sort* em JavaScript.

Figura 13 – Implementação do algoritmo *Selection Sort* em JavaScript

```

1  function selectionSort(vector) {
2      for (let i = 0; i < vector.length; i++) {
3          let min = i;
4
5          for (let j = i + 1; j < vector.length; j++)
6              if (vector[min] > vector[j]) min = j;
7
8          if (min !== i) {
9              let aux = vector[i];
10             vector[i] = vector[min];
11             vector[min] = aux;
12         }
13     }
14 }

```

Fonte: Jensen (2018) (Adaptado pelo Autor, 2021)

2.5 Desenvolvimento Móvel

O uso de dispositivos móveis, principalmente de *smartphones*, tem crescido consideravelmente, um exemplo disso é o resultado das pesquisas do IBGE, de 2019, que mostraram que 94% das residências brasileiras possuíam o aparelho e que das pessoas com mais de 10 anos, 81% possuíam os dispositivos para uso pessoal. Com essa crescente demanda, é possível perceber a necessidade de desenvolver aplicativos para estes dispositivos, de modo a atingir uma grande parte dos usuários.

Ao desenvolver um software, independente da plataforma em que ele será utilizado ou sua função, o desenvolvedor investe tempo, conhecimento e, conseqüentemente, dinheiro. Logo, o desenvolvimento de aplicativos para *smartphones* pode ser difícil para os desenvolvedores, pois eles precisam criar o mesmo aplicativo para diversos sistemas (SANTOS et al., 2016). Assim, o

desenvolvimento de aplicativos para esses dispositivos pode ser feito de forma híbrida ou de forma nativa.

Os aplicativos desenvolvidos de forma nativa para um sistema são aqueles sendo pensados para um tipo específico de plataforma (DA SILVA; SANTOS, 2014), ou seja, eles são feitos nas linguagens de programação utilizadas em seu sistema e, dificilmente, poderá utilizar o mesmo código-fonte para outro SO.

Para o SO Android, os *apps* nativos utilizam Java ou Kotlin como linguagem, enquanto os de iOS utilizam o Objective-C ou o Swift. Para programar nativamente para os dois principais sistemas operacionais de *smartphones* disponíveis no mercado, o custo com o desenvolvimento pode ser alto e quando é necessário realizar correções ou atualizações mais de um código-fonte deve ser revisado. Em contrapartida aos gastos e tempo de desenvolvimento, o desenvolvimento nativo sempre irá fornecer aplicativos com o melhor desempenho.

De modo a diminuir os gastos envolvidos na criação de aplicativos, algumas categorias de desenvolvimento foram criadas para permitir a construção híbrida de aplicativos, isto é, o desenvolvimento de um único código-fonte que serve para diversos sistemas.

Em comparação com o desenvolvimento nativo, dificilmente o desenvolvimento híbrido terá um desempenho superior. Entretanto, para aplicações que não necessitam de tanta performance, esse tipo de desenvolvimento se mostra uma boa opção, principalmente pelo menor custo de desenvolvimento.

Um dos tipos são os aplicativos híbridos são os *Web-Based*, feitos a partir de tecnologias utilizadas no desenvolvimento *web* (HTML, CSS e JS). Eles são baixados através da loja de aplicativos do sistema operacional, porém são exibidos através do *WebView* (ferramenta que permite a exibição de páginas web em aplicativos) do dispositivo (TAVARES, 2016). O *framework* Ionic é um exemplo de tecnologia que implementa essa categoria de desenvolvimento.

Há também o tipo *Cross-Compiled*, em que um único código-fonte é transformado para a linguagem nativa do sistema operacional em tempo de compilação, gerando assim, um aplicativo nativo (PONTES, 2017). Exemplos de tecnologias que utilizam essa categoria de desenvolvimento são o Xamarin e o Flutter.

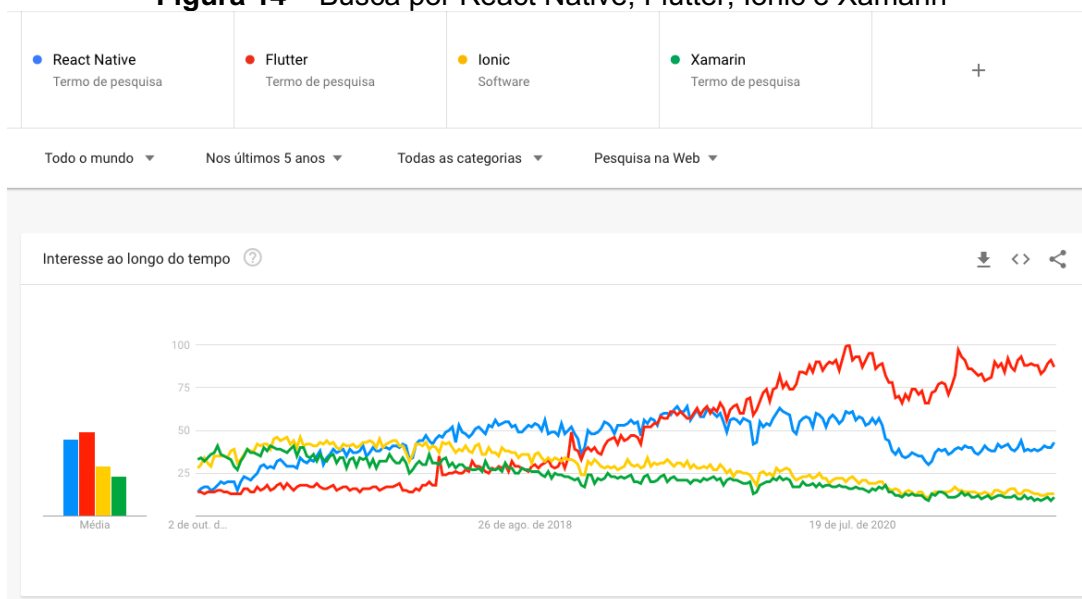
Por último, outra forma de desenvolvimento é o JavaScript *RunTime*, que tem seu código transformado em nativo em tempo de execução. Embora seja escrito em JS e transformado em *app* nativo durante a execução, os aplicativos desenvolvidos

nessa categoria acessam os recursos do sistema operacional através de pontes feitas em JavaScript (GUEDES, 2020). A principal tecnologia utilizada atualmente nessa categoria de desenvolvimento é o React Native.

2.6 Frameworks

Os *frameworks* são um conjunto de códigos especializados para uma finalidade (PONTES, 2017) e eles existem para as mais diversas necessidades, por exemplo, o Laravel para *backend* (escrito em PHP), o Bootstrap para o *frontend*, o Ionic, Xamarin, React Native e o Flutter para o desenvolvimento híbrido de apps. Os dois últimos são os *frameworks* de desenvolvimento híbrido mais pesquisados atualmente, como podemos ver na Figura 14, sendo que o Flutter lidera as pesquisas.

Figura 14 – Busca por React Native, Flutter, Ionic e Xamarin



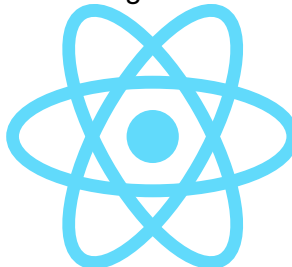
Fonte: Google Trends (2021)

2.6.1 React Native

O React Native, que a logo pode ser vista na Figura 15, foi anunciado em 2015 pelo Facebook e é um framework de código aberto utilizado para a criação de aplicativos, além de ser uma extensão do React, uma biblioteca para criação de interfaces do usuário (UI) que também foi criada pelo Facebook (LEITÃO, 2019). É escrito em JavaScript e é o *framework* de desenvolvimento híbrido baseado em JS

mais utilizado atualmente, sendo que empresas conhecidas, como Facebook, Skype, Instagram e Uber Eats o utilizam em suas aplicações (CANGUÇU, 2020).

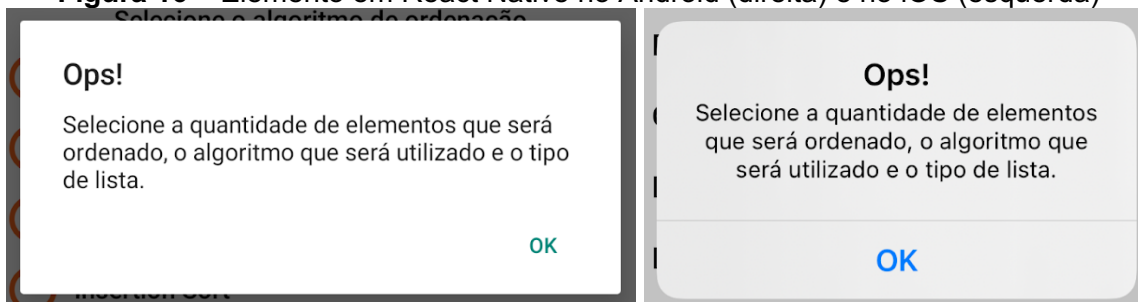
Figura 15 – Logo do React Native



Fonte: React Native (2021)

Os aplicativos criados com o React Native interpretam os componentes escritos em JS e os renderiza como os nativos do sistema (BEZERRA, 2021). Como vemos na Figura 16, os componentes poderão ter uma aparência no Android e outra no iOS, conforme as configurações do sistema.

Figura 16 – Elemento em React Native no Android (direita) e no iOS (esquerda)



Fonte: Autor (2021)

Outra característica do React Native é que ele funciona no formato JavaScript *RunTime*, ou seja, o código é compilado para a linguagem nativa do sistema em tempo de execução e os aplicativos acessam os recursos nativos através do próprio JavaScript.

2.6.2 Flutter

O Flutter é um *framework* de desenvolvimento híbrido baseado em Dart anunciado pela Google em 2015 e teve a primeira versão estável lançada em 2018 (BUENO, 2021). Ao criar um aplicativo utilizando o Flutter ele é compilado para a linguagem nativa do sistema operacional, fazendo com que ele possua acesso direto

aos recursos do dispositivo (DE ANDRADE, 2020). Sua logo pode ser vista na Figura 17.

Figura 17 – Logo do Flutter

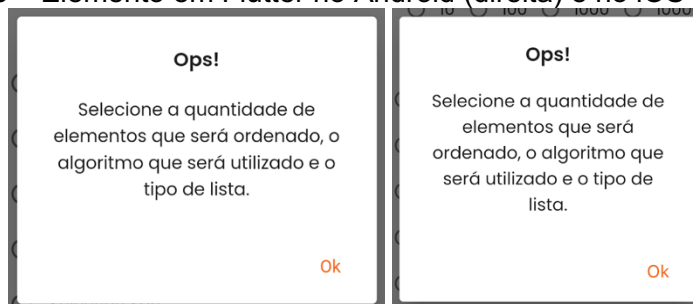


Fonte: Flutter (2021)

Segundo o site do próprio *framework*, empresas como o NuBank, Google e Alibaba utilizam o Flutter em suas aplicações. Uma característica do Flutter é que ele conta com *widgets* próprios, o que permite obter um design de interface personalizado (CANGUÇU, 2019).

Comparando um componente em sistemas operacionais diferentes, mas de um mesmo aplicativo desenvolvido em Flutter, podemos verificar na Figura 18 como a aparência dele é a mesma tanto no Android (posicionado à direita) quanto no iOS (posicionado à esquerda): mesmo estilo de borda, fonte etc.

Figura 18 – Elemento em Flutter no Android (direita) e no iOS (esquerda)



Fonte: Autor (2021)

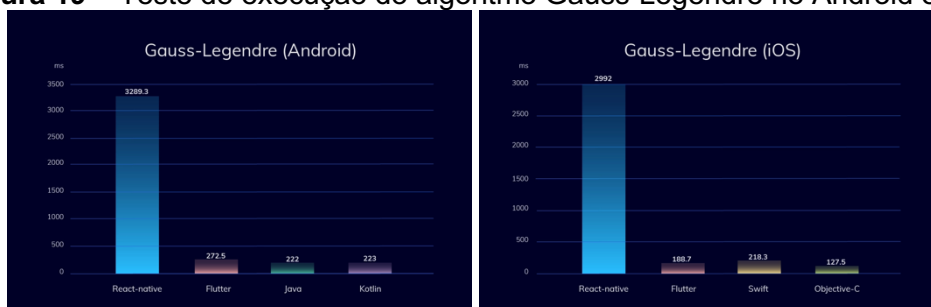
Outra característica do Flutter é que ele utiliza a categoria de desenvolvimento *Cross-Compiled*, onde o código é escrito em Dart e compilado para as linguagens nativas de cada SO.

2.7 Flutter vs. React Native

Por conta dos aplicativos desenvolvidos em Flutter serem transformados para a linguagem nativa do sistema em tempo de compilação, ele tende a obter os melhores resultados quando comparado a outros *frameworks* de desenvolvimento híbrido em relação ao desempenho.

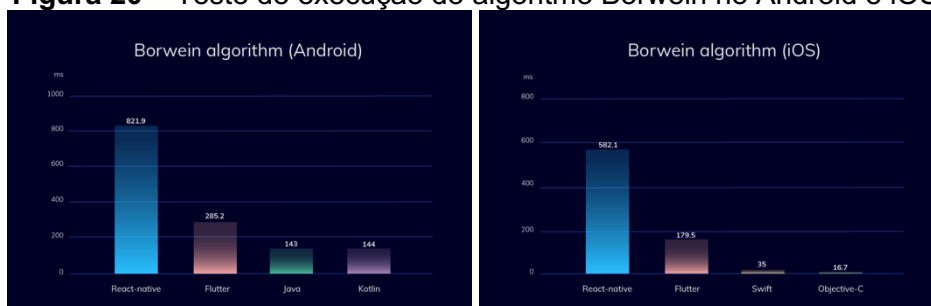
Demedyuk e Tsybulskyi (2020) avaliaram o desempenho do Flutter e do React ao executar algoritmos que exigem memória e processamento (CPU). A Figura 19 demonstra que o Flutter obteve um desempenho significativamente superior ao do React Native ao executar o algoritmo Gauss-Legendre, que exige memória, tanto no Android quanto no iOS. Ao rodar o algoritmo Borwein, que exige processamento, o maior desempenho também foi do Flutter em ambos os sistemas operacionais, como podemos ver na Figura 20.

Figura 19 – Teste de execução do algoritmo Gauss-Legendre no Android e iOS



Fonte: Demedyuk e Tsybulskyi (2020)

Figura 20 – Teste de execução do algoritmo Borwein no Android e iOS










Fonte: Demedyuk e Tsybulskyi (2020)

De acordo com Stender e Åkesson (2020), o Flutter também apresentou os melhores resultados em questão de CPU quando comparado com o React Native, sendo que o primeiro utilizou 2% a menos da CPU do que o segundo, entretanto apesar da vantagem no consumo de CPU, o Flutter demorou mais tempo para executar os códigos.

Além do desempenho, é importante também considerar fatores como curva de aprendizado e comunidade do *framework* durante o desenvolvimento. De Almeida (2019) determinou o Flutter como a tecnologia que apresentou o melhor resultado em sua aplicação teste após avaliar critérios como praticidade, documentação e curva de aprendizado, como podemos ver na Tabela 1.

Tabela 1 – Flutter vs. React Native de acordo com De Almeida (2019)

	A ferramenta que possui maior praticidade para ser utilizada?	A ferramenta que possui melhor documentação?	A ferramenta que tem menor curva de aprendizado inicial?	A ferramenta que entrega melhor experiência durante o desenvolvimento?	A ferramentas entregou o melhor resultado final?
					
					

Fonte: De Almeida (2019)

De acordo com Wu (2018), tanto o React Native quanto o Flutter se mostram boas opções de *frameworks* para o desenvolvimento híbrido de aplicações e ambos entregam eficiência e praticidade no desenvolvimento, mas o React Native, por conta da sua forte comunidade, pode ser definido como a melhor escolha para iniciar o desenvolvimento de um aplicativo.

Fentaw (2020) também destaca a comunidade como uma das principais vantagens de usar o React Native e que por usar o JavaScript torna-se de fácil aprendizado enquanto o Flutter, por utilizar Dart, tem uma pequena comunidade de desenvolvedores, o que pode dificultar o aprendizado.

Para Hjort (2020), quando comparado os dois *frameworks* em diversos critérios, como desempenho, plataformas, distribuição, viabilidade de longo prazo e outros, o React Native se mostrou superior, mas por pouca diferença entre eles, sendo as notas 4,5 atribuída ao React Native e 4,0 ao Flutter.

Da Silva (2021) também avaliou os dois *frameworks* levando em consideração fatores como tempo de preparação e de desenvolvimento, tamanho dos arquivos para distribuição, eficiência para acessar os recursos e outros, sendo que as notas ficaram iguais. Apesar desta igualdade, Silva (2021) destaca que o React Native obteve melhor resultado nas questões de viabilidade a longo prazo, o que demonstra sua

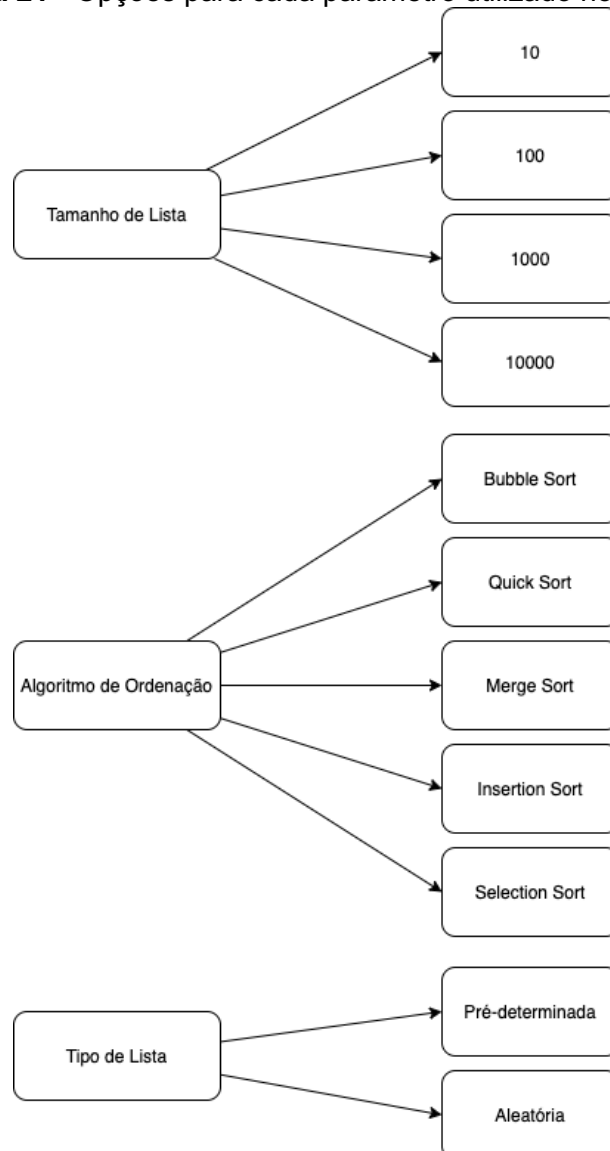
maturidade em relação ao Flutter, mesmo que este tenha obtido os melhores resultados em relação ao desempenho.

Com base nessas pesquisas, vemos então que embora o Flutter, na maioria das vezes, obtém um resultado superior em relação ao desempenho, o React Native se mostra uma forte tecnologia por conta de fatores como linguagem, comunidade e tempo de mercado.

3 METODOLOGIA

De modo a atingir os objetivos do trabalho, foram utilizadas duas aplicações, uma desenvolvida em Flutter e uma em React Native, onde o usuário pode escolher o tamanho da lista que seria utilizada, o algoritmo de ordenação que seria executado e o tipo de lista. Na Figura 21, é possível ver as opções disponíveis para cada parâmetro.

Figura 21 - Opções para cada parâmetro utilizado nos testes



Fonte: Autor (2021)

A ideia da lista pré-determinada é poder comparar os resultados dos aplicativos considerando um cenário idêntico ao outro, já que a ordem inicial da lista influencia nos resultados.

No desenvolvimento dos aplicativos só foram utilizados códigos dos *frameworks*, sem que alterações nos códigos nativos fossem feitas, para que a comparação levasse em conta também o que cada *framework* entrega.

Durante a execução foi contabilizado o tempo médio de execução e os dados obtidos foram analisados e comparados entre os aplicativos nos sistemas operacionais para *smartphones* iOS e Android. As especificações dos dispositivos utilizados para fazer os testes podem ser vistas na Tabela 2.

Tabela 2 – Especificação dos dispositivos utilizados para realizar os testes

Modelo	Marca	Processador	Memória RAM	SO
Poco X3 Pro	Xiaomi	Snapdragon 860 Qualcomm	6GB	Android 11
iPhone XR	Apple	Apple A12 Bionic	3GB	iOS 15

Fonte: Tudo Celular (2021) (Adaptado pelo Autor, 2021)

Embora existisse a ideia de medir também o consumo de memória RAM e de CPU, as bibliotecas disponíveis para tal só funcionavam para o Android e por conta disso foi decidido utilizar somente o tempo de execução.

3.1 Cenários

A pesquisa se deu em dois cenários possíveis, sendo o primeiro onde a lista já estava determinada, fazendo com que os resultados entre os dois *frameworks* tivessem uma comparação justa entre si. Já o segundo cenário é com o uso de uma lista gerada de forma aleatória, e, embora a comparação entre os dois *frameworks* não fique justa por conta da ordem da lista influenciar no resultado, é possível contabilizar o tempo que leva para gerar esta lista.

4 DESENVOLVIMENTO DO APLICATIVO

4.1 Ambiente de Desenvolvimento

O desenvolvimento do aplicativo foi feito utilizando um MacBook Air 2017 da Apple com processador Intel Core i5 Dual-Core 1.8 Ghz, 8GB de memória RAM, gráficos Intel HD Graphics 6000 e sistema operacional macOS Big Sur na versão 11.6. Além disso, o Visual Studio Code foi utilizado como editor de código-fonte e os aplicativos Android Studio e Xcode foram necessários para executar os simuladores durante o desenvolvimento.

Embora o CocoaPods, gerenciador de dependências utilizado tanto pelo Flutter quanto pelo React Native e o Homebrew, gerenciador de pacotes utilizado somente no último, já estivessem instalados, o resto da configuração dos *frameworks* foram feitas seguindo as suas respectivas documentações.

O Flutter foi instalado através do GitHub e foi necessário ajustar o arquivo de configuração do terminal do computador para que o comando flutter fosse reconhecido de qualquer pasta. Após o download dos arquivos, o comando *flutter doctor* foi executado para concluir a instalação, sendo que o tempo total utilizado para a configuração foi de 15 (quinze) minutos.

Já para o funcionamento do React Native foi necessário instalar o pacote Node, que permite a execução de códigos JavaScript fora do navegador, e o Watchman, ferramenta desenvolvida pelo Facebook que observa as mudanças no sistema de arquivos, otimizando o desempenho (REACT NATIVE, 2021).

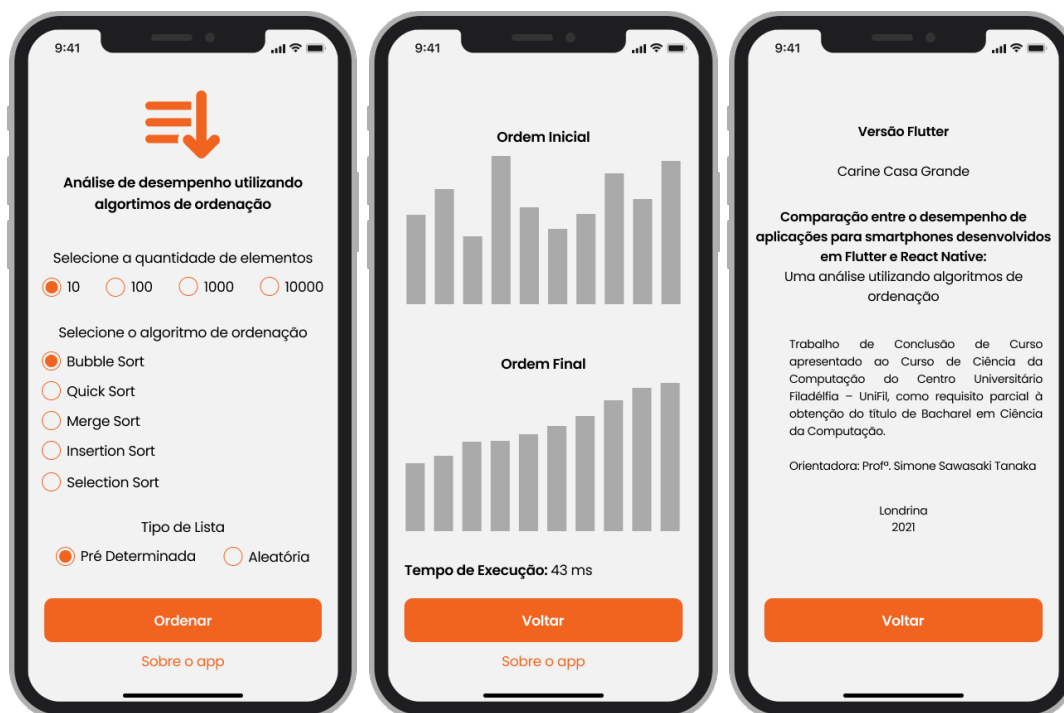
A instalação do Node foi rápida, entretanto a do Watchman falhou duas vezes sem motivo aparente ao baixar recursos dele. Além disso, após a criação do projeto foi baixado o pacote do React Native, fazendo com que o tempo total de instalação fosse de cerca de 50 (cinquenta) minutos.

4.2 Protótipo do Aplicativo

Desenhado no editor gráfico Figma, o *layout* do aplicativo é simples e tem como base de cor a logo da UniFil, como podemos ver na Figura 22. Na tela inicial, posicionada à direita, o usuário pode selecionar a quantidade de elementos que deseja ordenar, o algoritmo que será utilizado e se lista utilizada será uma pré-

determinada ou uma gerada aleatoriamente. Já na segunda tela, posicionada no centro, será ilustrada a ordem inicial e final da lista e o tempo de execução. Por último, a tela posicionada à direita da imagem, exibe apenas informações do aplicativo.

Figura 22 – Protótipo do aplicativo



Fonte: Autor (2021)

4.3 Desenvolvimento em Flutter

O desenvolvimento do aplicativo em Flutter demorou cerca de vinte horas, sendo que o desenvolvedor já possuía um conhecimento básico sobre o *framework*. Somente o pacote utilizado para gerar os gráficos precisou ser adicionado de forma manual ao projeto. Além disso, as fontes e imagens utilizadas no projeto tiveram que ser inseridas no arquivo de configuração do aplicativo, pois sem isso elas não eram reconhecidas.

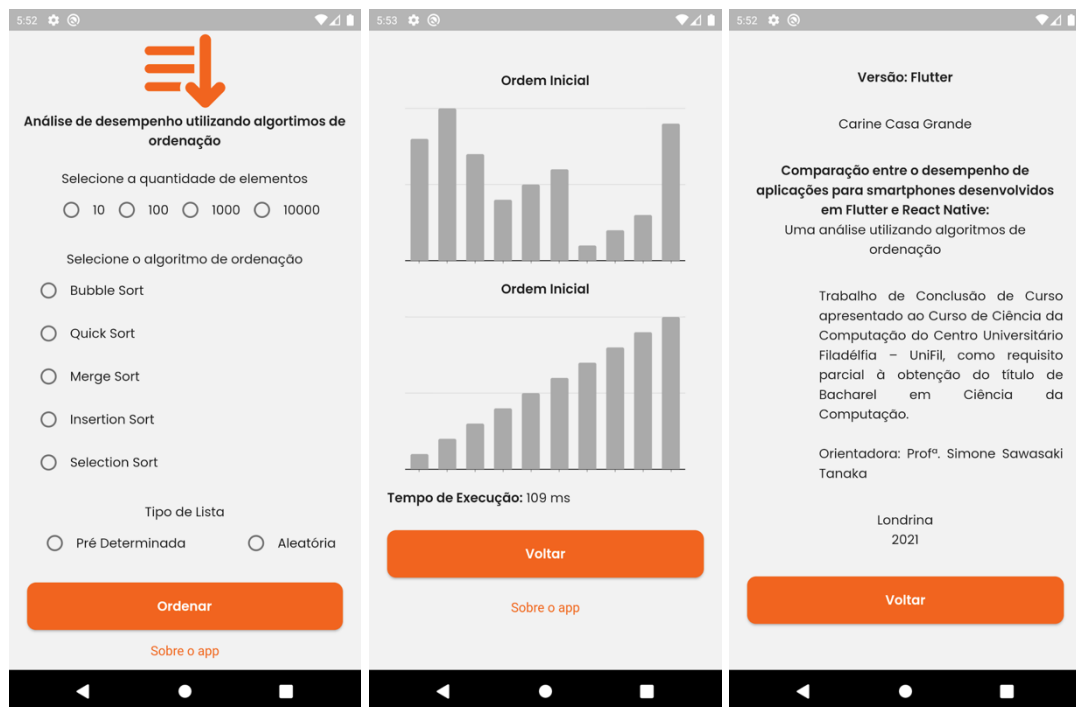
Todas as dúvidas que surgiram ao decorrer do desenvolvimento foram facilmente sanadas com as respostas encontradas em fóruns da comunidade, mesmo que eles não fossem tão grandes. Podemos ver na Figura 23 e na Figura 24 que o *layout* final do aplicativo ficou parecido com a ideia proposta pelo protótipo.

Figura 23 – Resultado do aplicativo construído em Flutter no iOS



Fonte: Autor (2021)

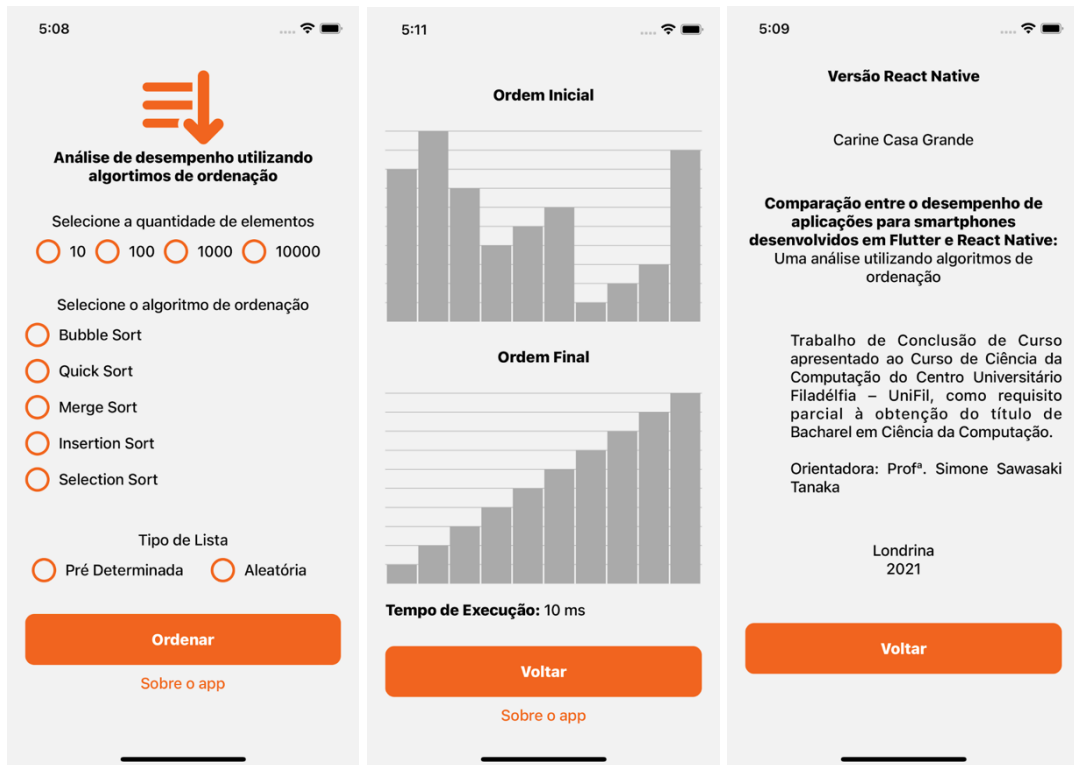
Figura 24 – Resultado do aplicativo construído em Flutter no Android



Fonte: Autor (2021)

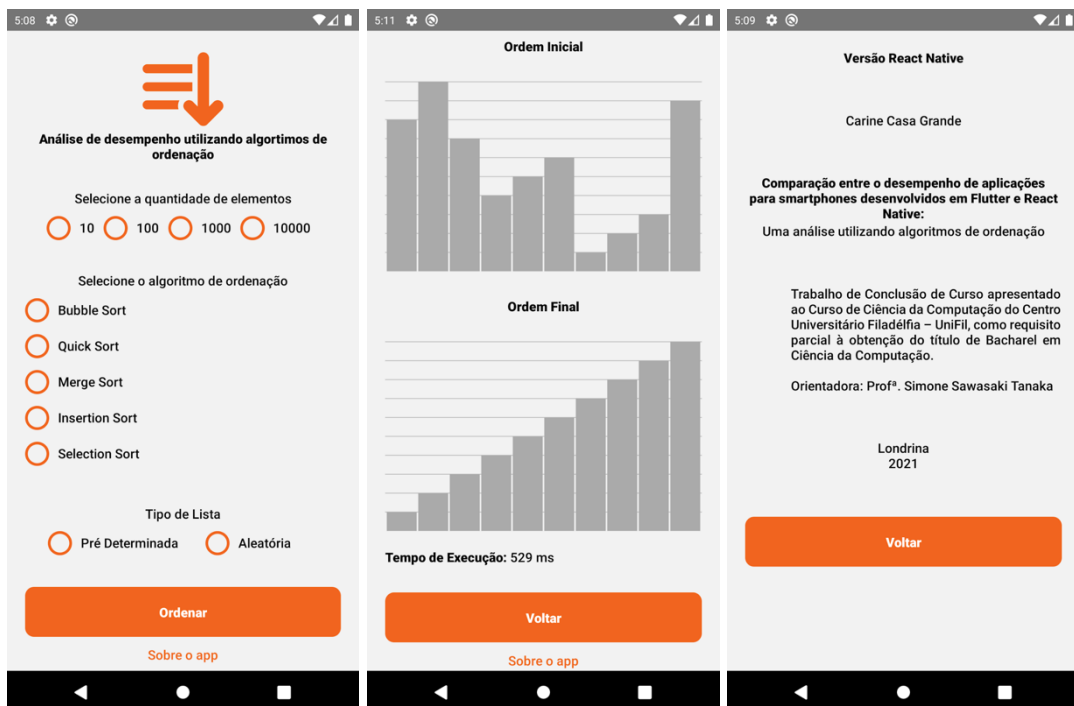
4.4 Desenvolvimento em React Native

Figura 25 – Resultado do aplicativo construído em React Native no iOS



Fonte: Autor (2021)

Figura 26 – Resultado do aplicativo construído em React Native no Android.



Fonte: Autor (2021)

O desenvolvimento do aplicativo em React Native também demorou cerca de vinte horas, sendo que, embora o desenvolvedor nunca tivesse usado o React ou o React Native, ele possuía bastante experiência com o desenvolvimento *web*.

Foi necessário baixar pacotes para navegação entre as telas, os Radio Buttons, que foram utilizados para seleção dos dados utilizados e um pacote para renderização dos gráficos.









Assim como no desenvolvimento do *app* em Flutter, todas as dúvidas foram resolvidas com as respostas encontradas em fóruns da comunidade. Podemos ver na Figura 25 e na Figura 26 que o *layout* final do aplicativo também ficou parecido com a ideia proposta pelo protótipo, exceto na questão da fonte utilizada.

5 RESULTADO E DISCUSSÃO

5.1 Android

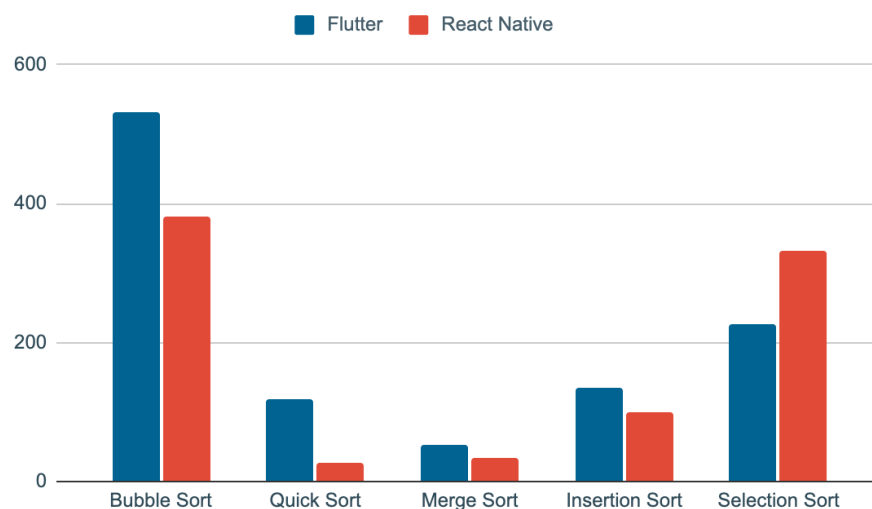
Na Tabela 3, vemos que o tempo que o Flutter leva para executar os algoritmos na lista pré-determinada com até 1000 (mil) elementos é consideravelmente menor que o tempo de execução do React Native, entretanto, ao ordenar a lista com 10000 (dez mil) elementos, o tempo de execução do React Native é menor em quase todos os algoritmos, exceto no Selection Sort.

Tabela 3 – Média de tempo utilizando a lista pré-determinada por quantidade de elementos

	10		100		1000		10000	
								
Bubble Sort	7,6	43,8	11,4	16,4	42,6	51	2062,6	1413,2
Quick Sort	6,8	36	8,4	15	11,8	20,8	446,4	37,6
Merge Sort	11,6	40,2	11,2	12,8	19	21,6	164,4	63,2
Insertion Sort	7,6	45,2	10,2	16,2	22,4	26,2	499,8	307
Selection Sort	8,8	42,2	8	16,4	25,6	46,2	860,2	1223,6

Fonte: Autor (2021)

Figura 27 – Média de tempo utilizando a lista pré-determinada por algoritmo de ordenação











Fonte: Autor (2021)

Ao analisarmos ao gráfico da Figura 27, vemos que o React Native se mostra a melhor opção na maior parte dos casos neste cenário, tendo levado menos tempo para executar em todos os algoritmos.

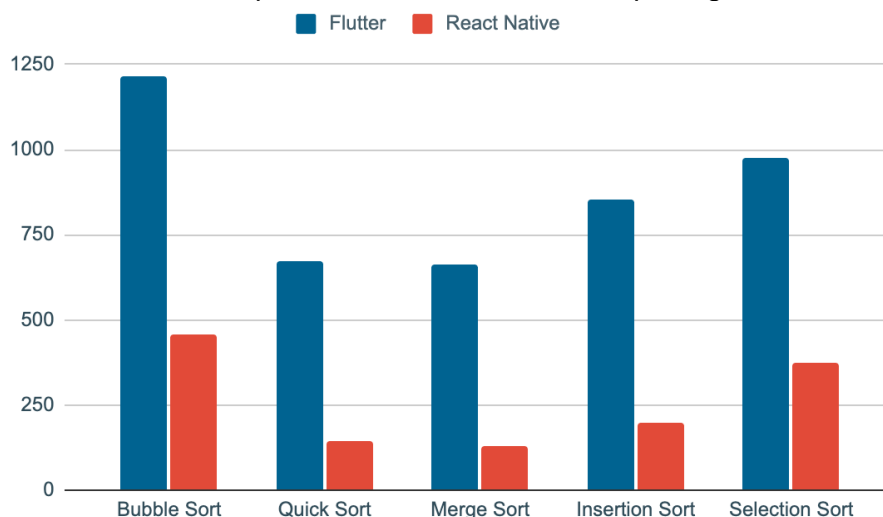
Assim como na lista pré-determinada, o React Native obteve os melhores resultados na ordenação das listas geradas aleatoriamente com mais elementos, como podemos ver na Tabela 4.

Tabela 4 – Média de tempo utilizando a lista aleatória por quantidade de elementos

	10		100		1000		10000	
								
Bubble Sort	10	41,8	8,6	17,6	67	53	4768,8	1722
Quick Sort	6,8	42,4	11,4	13,2	41,8	62	2623,6	463
Merge Sort	8,4	39,6	6,8	11,8	44,2	28,8	2592,2	442
Insertion Sort	10,4	42,4	9,8	14,4	50	44,4	3343,4	693,4
Selection Sort	11,2	11	6,4	12,4	47,4	39,8	3830,2	1431,2

Fonte: Autor (2021)

Figura 28 – Média de tempo utilizando a lista aleatória por algoritmo de ordenação











Fonte: Autor (2021)

Quando consideramos o tempo médio de execução por algoritmo de ordenação, como podemos ver na Figura 28, o React Native obteve um resultado menor do que o Flutter em todos os algoritmos, se mostrando novamente a melhor escolha para o cenário proposto.

Um outro ponto a ser considerado, é que o gráfico do Flutter, em todos os casos, demorou mais do que os gráficos do React Native para ser carregado, tornando a tela mais lenta.

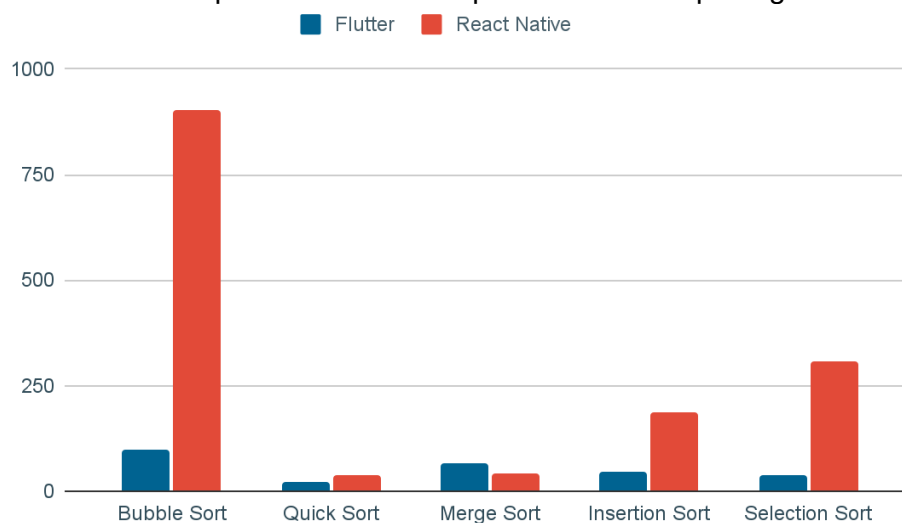
5.2 iOS

Tabela 5 – Média de tempo utilizando a lista pré-determinada por quantidade de elementos

	10		100		1000		10000	
								
Bubble Sort	34,4	48,4	19,6	28,6	16	71	319,6	3456,4
Quick Sort	18,8	18,8	15,6	29,6	20,6	28,4	32,4	79,6
Merge Sort	15,2	24,2	19,2	27,8	21,6	32,4	211,8	91,4
Insertion Sort	16,4	28,8	16,6	25,4	27,4	23	117,4	676,2
Selection Sort	17,2	21,8	19,6	24,6	18	29,8	94,8	1151,6

Fonte: Autor (2021)

Figura 29 – Média de tempo utilizando a lista pré-determinada por algoritmo de ordenação











Fonte: Autor (2021)

Ao comparar os resultados da lista pré-determinada no iOS, vemos que resultado do Flutter foi melhor do que o resultado do React Native na maior parte dos casos, como podemos ver na Tabela 5.

Considerando a média por algoritmo de ordenação, vemos na Figura 29 que o React Native levou menos tempo no Merge Sort, enquanto o flutter liderou os demais algoritmos de ordenação.

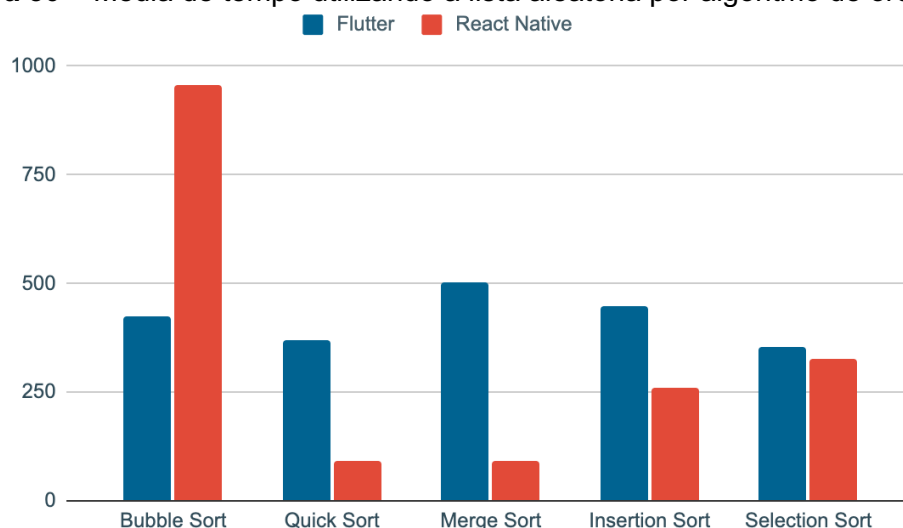
Ao analisarmos o resultado da lista aleatória vemos que nas listas com até 1000 (mil) elementos o Flutter obteve os melhores resultados, entretanto que na lista com 10000 (dez mil) elementos o React Native lidera com uma diferença grande entre os valores, como podemos ver na Figura 29.

Tabela 6 – Média de tempo utilizando a lista aleatória por quantidade de elementos

	10		100		1000		10000	
								
Bubble Sort	23,8	24	15	27	37	59,6	1624,2	3709,4
Quick Sort	15,2	28,6	16	22,2	32,4	31,8	1418,2	290,2
Merge Sort	13,4	27,8	16,4	25,6	49	24,8	1931,2	287,6
Insertion Sort	11,8	24	22,8	18,6	45,6	23,6	1711,8	981,8
Selection Sort	13,8	26,2	14,6	17,4	39,4	28,8	1347,8	1239

Fonte: Autor (2021)

Figura 30 – Média de tempo utilizando a lista aleatória por algoritmo de ordenação



Fonte: Autor (2021)

Na Figura 30, vemos que considerando os algoritmos de ordenação o Flutter obteve uma melhor média no Bubble Sort, enquanto o React Native teve uma melhor média nos demais algoritmos.

Assim como ocorreu no Android, os gráficos demoraram mais para carregar no aplicativo desenvolvido em Flutter do que o desenvolvido em React Native, mas como se trata de bibliotecas de terceiros,

5.2.3 Análise

Embora os estudos correlatos indicassem que o Flutter teria um resultado superior do que o React Native, isto não ocorreu. Para as listas com menos elementos o Flutter executou de forma mais rápida os algoritmos, mas para as listas maiores, que exigem maior capacidade de processamento e memória, o React Native foi mais ágil.

6 CONCLUSÃO

Escolher corretamente qual tecnologia será utilizada no desenvolvimento de um *software* é de extrema importância para fazer com o mesmo ofereça uma boa experiência para o usuário. Este trabalho comparou o desempenho do Flutter e do React Native ao executar alguns algoritmos de ordenação, tendo como hipótese a ideia de que o Flutter teria os melhores resultados por conta dos trabalhos correlatos que existem na área.

Entretanto, os testes apontaram que o React Native obteve os melhores resultados em 75% dos cenários testados, tendo executado os algoritmos em um tempo menor do que o Flutter. Além disso, o carregamento da página que o React Native forneceu foi inferior ao do Flutter, tornando o aplicativo mais fluido.

O fato de o Flutter ter obtido um resultado inferior não o torna uma tecnologia ruim, sendo que tudo depende da necessidade do aplicativo que será desenvolvido. O React Native se mostra uma tecnologia muito forte, tanto pelo seu desempenho quanto pela facilidade que é aprender a tecnologia, entretanto muitos elementos devem ser importados, o que pode dificultar o desenvolvimento.

REFERÊNCIAS

- ASCENCIO, Ana Fernanda Gomes; DE ARAÚJO, Graziela Santos. **Estrutura de Dados**: algoritmos, análise da complexidade e implementações em Java e C/C++. 1 ed. São Paulo, SP: Editora Pearson, 2010. 250 p. (ISBN: 9788576058816).
- BEZERRA, Franklyn Seabra Rogério. **Desenvolvimento Nativo vs Ionic vs React Native**: uma análise comparativa do suporte à acessibilidade em Android. Fortaleza, CE, f. 25, 2021. Trabalho de Conclusão de Curso (Computação) - Universidade Federal do Ceará.
- BORIN, Vinicius Pozzobon. **Estrutura de dados**. 1 ed. Curitiba, PR: Editora Contentus, 2020. 178 p. (ISBN: 9786557451595).
- BUENO, Carlos Eduardo de Oliveira. **Desenvolvimento de um aplicativo utilizando o framework Flutter e arquitetura limpa**. Goiânia, GO, f. 47, 2021. Trabalho de Conclusão de Curso (Ciência da Computação) - Pontifícia Universidade Católica de Goiás.
- CANGUÇU, Raphael. **React Native no Desenvolvimento de Aplicativos Famosos**. 2020. Disponível em: <https://codificar.com.br/react-native-no-desenvolvimento-de-aplicativos-famosos/>. Acesso em: 22 set. 2021.
- CANGUÇU, Raphael. **React Native vs. Flutter**. 2019. Disponível em: <https://codificar.com.br/react-native-vs-flutter/>. Acesso em: 22 set. 2021.
- CORMEN, Thomas H. et al. **Algoritmos**: Teoria e Prática (Tradução da 3ª Edição Americana). Rio de Janeiro, RJ: Editora Elsevier, 2012. 1042 p. (ISBN 978-85-352-3699-6).
- DA FONSECA, Diego Batista et al. Comparativo de desempenho na execução entre Linguagens de Programação. *In*: XIII WORKSHOP DE COMPUTAÇÃO DA FACCAMP. 2017. Anais [...] Campo Limpo Paulista, SP, 2017. Disponível em: http://www.cc.faccamp.br/anaisdowcf/edicao_atual/wcf2017/wcf2017.pdf. Acesso em: 26 set. 2021.
- DA ROSA, Luciano; ANTONIAZZI, Rodrigo Luiz. Métodos de Ordenação de Dados: Uma Análise Prática I. *In*: XVII SEMINÁRIO INTERINSTITUCIONAL DE ENSINO, PESQUISA E EXTENSÃO; XV MOSTRA DE INICIAÇÃO CIENTÍFICA E X MOSTRA DE EXTENSÃO. 2012. Anais [...] Cruz Alta, RS, 2012. Disponível em: <https://home.unicruz.edu.br/seminario/anais/anais-2012/ccaet/metodos%20de%20ordenacao%20de%20dados%20uma%20analise%20pratica%20i.pdf>. Acesso em: 22 set. 2021.
- DA SILVA, Antônio Eudálio de Sousa. **Análise comparativa entre os frameworks de desenvolvimento de aplicativos móveis multiplataforma**. Quixadá, CE, 2021. Trabalho de Conclusão de Curso (Sistema de Informação) - Universidade Federal

do Ceará. Disponível em: <http://repositorio.ufc.br/handle/riufc/59037>. Acesso em: 28 set. 2021.

DA SILVA, Marcelo Moro; SANTOS, Marilde Terezinha Prado. Os Paradigmas de Desenvolvimento de Aplicativos para Aparelhos Celulares. **Revista TIS**, [São Carlos, SP], v. 3, n. 2, p. 162-170, 2014. Disponível em: <http://revistatis.dc.ufscar.br/index.php/revista/article/view/86>. Acesso em: 22 set. 2021.

DE ALMEIDA, Robson Rosa. **Tecnologias para o desenvolvimento de aplicações multiplataforma**: um estudo sobre os frameworks React Native e Flutter. Porto Alegre, RS, f. 74, 2019. Trabalho de Disciplina (Curso Superior de Tecnologia em Sistemas para Internet) - Faculdade e Escola Técnica Alcides Maya. Disponível em: <http://raam.alcidesmaya.com.br/index.php/projetos/article/view/54>. Acesso em: 27 set. 2021.

DE ANDRADE, Ana Paula. **O que é Flutter?**. 2020. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-flutter>. Acesso em: 22 set. 2021.

DEMEDYUK, Ihor; TSYBULSKYI, Nazar. **Flutter vs Native vs React-Native: Examining Performance**. Disponível em: <https://inveritasoft.com/blog/flutter-vs-native-vs-react-native-examining-performance>. Acesso em: 22 set. 2021.

DIAS, Andrew Carlos de Sene; VILELA, Nayara Almeida; JÚNIOR, Walteno Martins Parreira. Análise sobre alguns métodos de ordenação de listas: seleção, inserção e shellsort. **Intercursos Revista Científica**, Ituiutaba, MG, v. 13, n. 1, p. 64-76, 2014. Disponível em: <https://revista.uemg.br/index.php/intercursosrevistacientifica/article/view/2452>. Acesso em: 26 set. 2021.

DOS SANTOS, Ilso Francisco. **Algoritmos de Ordenação**: uma abordagem didática para o Ensino Médio. Recife, PE, f. 79, 2020. Dissertação (Matemática) - Universidade Federal Rural de Pernambuco.

FARIAS, Fábio Henrique de; DA SILVA, Fabiano Barbosa Mendes; NETO, José Belmiro. Quicksort e Quicksort Aleatorizado: Um estudo comparativo. In: CONGRESSO DE MATEMÁTICA APLICADA E COMPUTACIONAL. 2012. Anais [...] Garanhuns, PE, 2012, p. 88-90. Disponível em: <http://arquivo.sbmec.org.br/cmecs/cmec-ne/2012/trabalhos/>. Acesso em: 26 set. 2021.

FEIJÓ, Valéria Casaroto; GONÇALVES, Berenice Santos; GOMEZ, Luiz Salomão Ribas. Heurística para avaliação de usabilidade em interfaces de aplicativos smartphones: utilidade, produtividade e imersão. **Design & Tecnologia**, Florianópolis, SC, v. 3, n. 6, p. 33-42, 31 dez. 2013. Disponível em: <https://doi.org/10.23972/det2013iss06pp33-42>. Acesso em: 21 set. 2021.

FENTAW, Awel Eshetu. **Cross platform mobile application development: a comparison study of React Native Vs Flutter**. Jyväskylä, Finlândia. Tese (Tecnologia da Informação) - University Of Jyväskylä. Disponível em: <http://urn.fi/URN:NBN:fi:ju-202006295155>. Acesso em: 27 set. 2021.

FIGUEIREDO, Carlos Maurício Seródio; NAKAMURA, Eduardo. Computação Móvel: Novas Oportunidades e Novos Desafios. **T&C Amazônia**, Manaus, AM, v. 1, n. 2, p. 16-28, jun. 2003. Disponível em: <https://www.researchgate.net/publication/268435975>. Acesso em: 21 set. 2021.

FLUTTER. **Design Beautiful Apps**. 2021. Disponível em: <https://flutter.dev/>. Acesso em: 29 set. 2021.

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. **Lógica de programação: a construção de algoritmos e estruturas de dados**. 3 ed. São Paulo, SP: Editora Pearson, 2005. 232 p. (ISBN: 9788576050247).

GEEKSFORGEEKS. **Insertion Sort**. 2021. Disponível em: <https://www.geeksforgeeks.org/insertion-sort/>. Acesso em: 27 set. 2021.

GOOGLE TRENDS. **Interesse ao longo do tempo**: Bubble sort, Quicksort, Merge sort, Insertion sort e Selection sort. Disponível em: https://trends.google.com.br/trends/explore?date=today%205-y&q=%2Fm%2F01ck9,%2Fm%2F092crt,%2Fm%2F050_s,%2Fm%2F03xsl,%2Fm%2F078bg. Acesso em: 22 set. 2021.

GOOGLE TRENDS. **Interesse ao longo do tempo**: React Native, Flutter, Ionic e Xamarin. Disponível em: https://trends.google.com.br/trends/explore?date=today%205-y&q=React%20Native,Flutter,%2Fg%2F1q6l_n0n0,Xamarin. Acesso em: 27 set. 2021.

GOOGLE TRENDS. **Interesse ao longo do tempo**: Selection sort, Heapsort, Shell sort, Bogosort e Timsort. Disponível em: <https://trends.google.com.br/trends/explore?date=today%205-y&q=%2Fm%2F078bg,%2Fm%2F03mwc,%2Fm%2F0k8m7,%2Fm%2F0pgn7,%2Fm%2F076wtj2>. Acesso em: 22 set. 2021.

GUEDES, Marylene. **O que é Dart?**. 2020. Disponível em: <https://www.treinaweb.com.br/blog/o-que-e-dart>. Acesso em: 21 set. 2021.

GUEDES, Marylene. **React Native ou Flutter: por qual começar?**. 2020. Disponível em: <https://www.treinaweb.com.br/blog/react-native-ou-flutter-por-qual-comecar/>. Acesso em: 22 set. 2021.

GUEDES, Sérgio (Org.). **Lógica de Programação Algorítmica**. 1 ed. São Paulo, SP: Editora Pearson, 2014. 160 p. (ISBN: 9788543005546).

GUIMARÃES, Davi. **Algoritmos de ordenação**: análise e comparação. 2014. Disponível em: http://davitpd.blogspot.com/2014/07/algoritmos-de-ordenacao-analise-e_13.html. Acesso em: 26 set. 2021.

GUPTA, Vikram. **Visualizing, Designing, and Analyzing the Merge Sort Algorithm**. 2020. Disponível em: <https://levelup.gitconnected.com/visualizing-designing-and-analyzing-the-merge-sort-algorithm-cf17e3f0371f>. Acesso em: 26 set. 2021.

HJORT, Elin. **Evaluation of React Native and Flutter for cross-platform mobile application development**. Turku, Finlândia, 2020. Tese (Engenharia da Computação) - Universidade Åbo Akademi. Disponível em: <https://www.doria.fi/handle/10024/180002>. Acesso em: 28 set. 2021.

HONORATO, Bruno de Almeida. **Algoritmos de ordenação**: análise e comparação. 2013. Disponível em: <https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261>. Acesso em: 21 set. 2021.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA. **Uso de Internet, televisão e celular no Brasil**. Disponível em: <https://educa.ibge.gov.br/jovens/materias-especiais/20787>. Acesso em: 21 set. 2021.

JENSEN, Kyle. **Javascript Algorithms**: Selection Sort. 2018. Disponível em: <https://medium.com/javascript-algorithms/javascript-algorithms-selection-sort-54da919d0513>. Acesso em: 27 set. 2021.

JÚNIOR, Paulo Afonso Parreira. **Estrutura de dados**: Primeiros passos com métodos de busca. 2015. Disponível em: <https://www.devmedia.com.br/estrutura-de-dados-primeiros-passos-com-metodos-de-busca/33018>. Acesso em: 21 set. 2021.

LEITÃO, Paulo Rogério Matheus Bonfim. **Desenvolvimento nativo vs React Native**: uma análise comparativa na codificação de uma aplicação para fitness. Fortaleza, CE, f. 54, 2019 Trabalho de Conclusão de Curso (Sistemas e Mídias Digitais) - Universidade Federal do Ceará.

LICHAO, Wang. **Javascript recursive quicksort**. 2014. Disponível em: <https://gist.github.com/0532/436b6c5cd2317a45ec3e>. Acesso em: 23 set. 2021.

MENDONÇA, Vinícius Rafael Lobo; BITTAR, Thiago Jabur; DIAS, Márcio de Souza. Um estudo dos Sistemas Operacionais Android e iOS para o desenvolvimento de aplicativos. *In*: ENCONTRO ANUAL DE COMPUTAÇÃO. 2011. Anais [...] Catalão, GO, 2011. Disponível em: <https://www.enacomp.com.br/biblioteca-de-publicacoes/publication-details.php?id=59>. Acesso em: 21 set. 2021.

MOZILLA FOUNDATION. **Sobre JavaScript**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/About_JavaScript. Acesso em: 21 set. 2021.

NEHRA, Pulkit. **Quicksort vs Merge Sort Which is better?**. 2020. Disponível em: <https://medium.com/@pulkitnehra/quicksort-vs-merge-sort-which-is-better-6eb208ab27c5>. Acesso em: 23 set. 2021.

PEREIRA, Ronaldo dos Santos. **Controle e aquisição de dados experimentais com tecnologia bluetooth em dispositivos móveis**. Ilha Solteira, SP, 2016. 113 p. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Estadual Paulista – Unesp. Disponível em: <http://hdl.handle.net/11449/138021>. Acesso em: 21 set. 2021.

PEREIRA, Wilder. **Introdução à Complexidade de Algoritmos**. 2019. Disponível em: <https://medium.com/nagoya-foundation/introdu%C3%A7%C3%A3o-%C3%A0-complexidade-de-algoritmos-4a9c237e4ecc>. Acesso em: 22 set. 2021.

PONTES, Stefano Walker Pereira. **Desenvolvimento de aplicativos móveis híbridos: um estudo de caso sobre o rMiase**. São Luís, MA, f. 82, 2017. Trabalho de Conclusão de Curso (Ciência da Computação) - Universidade Federal do Maranhão.

PROGRAMIZ. **Bubble Sort**. Disponível em: <https://www.programiz.com/dsa/bubble-sort>. Acesso em: 22 set. 2021.

PROGRAMIZ. **Merge Sort Algorithm**. Disponível em: <https://www.programiz.com/dsa/merge-sort>. Acesso em: 24 set. 2021.

REACT NATIVE. **Learn once, write anywhere**. 2021. Disponível em: <https://reactnative.dev/>. Acesso em: 29 set. 2021.

ROVEDA, Ugo. **Linguagem de programação: o que é e qual linguagem aprender**. 2019. Disponível em: <https://kenzie.com.br/blog/linguagem-de-programacao/>. Acesso em: 21 set. 2021.

SALUTES, Bruno. **iOS: cinco curiosidades sobre o sistema da Apple**. 2019. Disponível em: <https://canaltech.com.br/ios/ios-cinco-curiosidades-sobre-o-sistema-da-apple/>. Acesso em: 21 set. 2021.

SANTOS, Alan R. et al. Investigating the Adoption of Agile Practices in Mobile Application Development. *In: 18TH INTERNATIONAL CONFERENCE ON ENTERPRISE INFORMATION SYSTEMS*. 2016, [Porto Alegre, RS], 2016, p. 490-497. Disponível em: <https://www.researchgate.net/publication/298759850>. Acesso em: 22 set. 2021.

SARTORELI, Carlos Eduardo; UNO, Nairemilia KUCHAUSKI Alves. Comparativo entre iOS, Android e Windows Phone. *In: ETIC - ENCONTRO DE INICIAÇÃO*

CIENTÍFICA. 2013. Anais [...] Presidente Prudente, SP, 2013. Disponível em: <http://intertemas.toledoprudente.edu.br/index.php/ETIC/issue/view/60>. Acesso em: 21 set. 2021.

SILVA, Victor Augusto Fernandes da. **Revisão dos Algoritmos de Ordenação**. Passos, MG, 2018. Trabalho de Disciplina - Instituto Federal da Ciência e Tecnologia do Sul de Minas Gerais. Disponível em: <https://www.researchgate.net/publication/325050627>. Acesso em: 26 set. 2021.

SOUZA, Jackson E. G.; RICARTE, João Victor G.; LIMA, Náthalee Cavalcanti de Almeida. Algoritmos de Ordenação: Um Estudo Comparativo. *In: II ENCONTRO DE COMPUTAÇÃO DO OESTE POTIGUAR*. Anais [...] Pau dos Ferros, RN, 2017. Disponível em: <https://periodicos.ufersa.edu.br/index.php/ecop/article/view/7082>. Acesso em: 23 set. 2021.

STATCOUNTER. **Mobile Operating System Market Share Brazil**: Jan - Dec 2020. Disponível em: <https://gs.statcounter.com/os-market-share/mobile/brazil/2020>. Acesso em: 21 set. 2021.

STATCOUNTER. **Mobile Operating System Market Share Worldwide**: Jan - Dec 2020. Disponível em: <https://gs.statcounter.com/os-market-share/mobile/worldwide/2020>. Acesso em: 21 set. 2021.

STATCOUNTER. **Operating System Market Share Worldwide**: Jan - Dec 2020. Disponível em: <https://gs.statcounter.com/os-market-share/all/worldwide/2020>. Acesso em: 21 set. 2021.

STENDER, Simon; ÅKESSON, Hampus. **Cross-platform Framework Comparison**: Flutter & React Native. Karlskrona, Suécia, f. 47, 2020. Tese (Engenharia de Software) - Instituto de Tecnologia de Blekinge.

TAVARES, Henrique Leal. Introdução ao desenvolvimento de aplicações híbridas. **Revista eletrônica da Fatec Garça**, Garça, SP, v. 6, 2016. Disponível em: <https://fatecgarca.edu.br/ojs/index.php/efatec/article/view/113>. Acesso em: 22 set. 2021.

TUDO CELULAR. **iPhone XR**: Ficha Técnica. 2021. Disponível em: <https://www.tudocelular.com/Apple/fichas-tecnicas/n4820/Apple-iPhone-XR.html>. Acesso em: 5 out. 2021.

TUDO CELULAR. **Poco X3 Pro**: Ficha Técnica. 2021. Disponível em: <https://www.tudocelular.com/Poco/fichas-tecnicas/n6681/Poco-X3-Pro.html>. Acesso em: 5 out. 2021.

VEENSTRA, Brianna. **Merge Sort**. 2016. Disponível em: <https://github.com/sf-wdi-31/mergesort>. Acesso em: 26 set. 2021.

WU, Wenhao. **React Native vs Flutter, cross-platform mobile application frameworks**. Helsinque, Finlândia, f. 34, 2018. Tese (Engenharia) - Metropolia University Of Applied Sciences. Disponível em: <https://www.theseus.fi/bitstream/handle/10024/146232/thesis.pdf>. Acesso em: 27 set. 2021.

ZIVIANI, Nivio. **Projeto de algoritmos**: com implementações em Java e C+. 4 ed. São Paulo, SP: Editora Pioneira, f. 311, 1999. 621 p.

APÊNDICES

Apêndice A – Teste do Flutter realizado com lista pré-determinada no iOS.

		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Bubble Sort	10	66	22	37	25	22	34,4
	100	29	12	11	22	24	19,6
	1000	14	15	18	18	15	16
	10000	322	327	309	312	328	319,6
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Quick Sort	10	28	17	16	17	16	18,8
	100	15	13	12	16	22	15,6
	1000	16	24	21	26	16	20,6
	10000	47	47	32	16	20	32,4
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Merge Sort	10	26	15	9	14	12	15,2
	100	14	14	23	23	22	19,2
	1000	37	21	16	15	19	21,6
	10000	226	215	206	205	207	211,8
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Insertion Sort	10	19	21	15	19	8	16,4
	100	17	22	16	16	12	16,6
	1000	21	16	58	9	33	27,4
	10000	129	112	109	127	110	117,4
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Selection Sort	10	22	16	17	13	18	17,2
	100	20	16	16	24	22	19,6
	1000	22	23	14	15	16	18
	10000	93	85	72	116	108	94,8

Apêndice B – Teste do Flutter realizado com lista pré-determinada no Android.

		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Bubble Sort	10	2	14	5	3	14	7,6
	100	13	13	8	9	14	11,4
	1000	47	38	43	34	51	42,6
	10000	2076	2053	2080	2055	2049	2062,6
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Quick Sort	10	8	3	8	11	4	6,8
	100	3	5	14	8	12	8,4
	1000	12	10	14	8	15	11,8
	10000	40	26	51	49	2066	446,4
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Merge Sort	10	6	10	14	13	15	11,6
	100	15	12	6	12	11	11,2
	1000	36	21	9	21	8	19
	10000	274	274	238	27	9	164,4
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Insertion Sort	10	2	14	10	2	10	7,6
	100	8	14	13	3	13	10,2
	1000	27	31	17	11	26	22,4
	10000	505	504	473	512	505	499,8
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Selection Sort	10	11	12	9	9	3	8,8
	100	6	12	10	8	4	8
	1000	37	43	24	11	13	25,6
	10000	865	848	848	857	883	860,2

Apêndice C – Teste do React Native realizado com lista pré-determinada no iOS.

		1ª	2ª	3ª	4ª	5ª	Tempo
		Execução	Execução	Execução	Execução	Execução	Médio
Bubble Sort	10	129	27	29	28	29	48,4
	100	29	34	30	24	26	28,6
	1000	156	36	38	76	49	71
	10000	3061	3682	3337	3512	3690	3456,4
		1ª	2ª	3ª	4ª	5ª	Tempo
		Execução	Execução	Execução	Execução	Execução	Médio
Quick Sort	10	30	28	5	3	28	18,8
	100	29	33	29	28	29	29,6
	1000	34	51	10	39	8	28,4
	10000	82	83	83	80	70	79,6
		1ª	2ª	3ª	4ª	5ª	Tempo
		Execução	Execução	Execução	Execução	Execução	Médio
Merge Sort	10	38	26	29	22	6	24,2
	100	24	28	27	30	30	27,8
	1000	35	38	36	16	37	32,4
	10000	97	91	89	87	93	91,4
		1ª	2ª	3ª	4ª	5ª	Tempo
		Execução	Execução	Execução	Execução	Execução	Médio
Insertion Sort	10	43	26	27	28	20	28,8
	100	36	27	30	14	20	25,4
	1000	40	15	14	18	28	23
	10000	666	673	696	667	679	676,2
		1ª	2ª	3ª	4ª	5ª	Tempo
		Execução	Execução	Execução	Execução	Execução	Médio
Selection Sort	10	5	24	28	24	28	21,8
	100	30	30	4	31	28	24,6
	1000	49	16	46	17	21	29,8
	10000	1004	1065	1406	1217	1066	1151,6

Apêndice D – Teste do React Native realizado com lista pré-determinada no Android.

		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Bubble Sort	10	172	4	14	11	18	43,8
	100	21	11	15	18	17	16,4
	1000	55	47	39	61	53	51
	10000	1271	1302	1471	1497	1525	1413,2
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Quick Sort	10	132	13	14	9	12	36
	100	22	16	16	9	12	15
	1000	26	21	21	18	18	20,8
	10000	63	25	43	17	40	37,6
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Merge Sort	10	146	10	15	12	18	40,2
	100	12	15	14	12	11	12,8
	1000	18	34	20	17	19	21,6
	10000	90	57	57	53	59	63,2
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Insertion Sort	10	172	10	11	20	15	45,6
	100	20	17	13	14	17	16,2
	1000	30	23	35	21	22	26,2
	10000	306	297	298	303	331	307
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Selection Sort	10	160	11	16	13	11	42,2
	100	20	18	18	15	11	16,4
	1000	42	46	36	42	65	46,2
	10000	1254	1235	1241	1224	1164	1223,6

Apêndice E – Teste do Flutter realizado com lista aleatória no iOS.

		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Bubble Sort	10	30	23	19	19	28	23,8
	100	16	11	13	14	21	15
	1000	50	40	18	31	46	37
	10000	1614	1736	1492	1612	1667	1624,2
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Quick Sort	10	21	17	11	18	9	15,2
	100	8	23	16	12	21	16
	1000	31	25	30	32	44	32,4
	10000	1906	1229	1249	1487	1220	1418,2
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Merge Sort	10	31	11	7	11	7	13,4
	100	21	11	20	20	10	16,4
	1000	50	56	44	41	54	49
	10000	1426	1501	3599	1531	1599	1931,2
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Insertion Sort	10	17	10	17	8	7	11,8
	100	14	6	67	17	10	22,8
	1000	39	30	65	43	51	45,6
	10000	1434	1982	2258	1292	1593	1711,8
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Selection Sort	10	19	16	17	7	10	13,8
	100	18	15	8	14	18	14,6
	1000	44	29	41	30	53	39,4
	10000	1464	1245	1428	1271	1331	1347,8

Apêndice F – Teste do Flutter realizado com lista aleatória no Android.

		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Bubble Sort	10	2	1	17	16	14	10
	100	11	5	12	9	6	8,6
	1000	68	56	60	64	87	67
	10000	4953	4655	4810	4661	4765	4768,8
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Quick Sort	10	4	6	7	14	3	6,8
	100	7	17	12	6	15	11,4
	1000	45	35	40	51	38	41,8
	10000	2642	2563	2778	2413	2722	2623,6
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Merge Sort	10	4	10	12	15	1	8,4
	100	8	9	7	5	5	6,8
	1000	58	48	45	42	28	44,2
	10000	3127	2761	2820	4187	66	2592,2
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Insertion Sort	10	13	8	11	8	12	10,4
	100	10	16	6	12	5	9,8
	1000	52	67	45	35	51	50
	10000	3193	3340	3667	2999	3518	3343,4
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Selection Sort	10	13	4	12	13	14	11,2
	100	4	6	13	6	3	6,4
	1000	49	78	38	33	39	47,4
	10000	3579	3809	3459	3698	4606	3830,2

Apêndice G – Teste do React Native realizado com lista aleatória no iOS.

		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Bubble Sort	10	39	26	29	4	22	24
	100	21	26	30	31	27	27
	1000	82	40	41	55	80	59,6
	10000	3398	3364	3785	3939	4061	3709,4
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Quick Sort	10	40	22	28	28	25	28,6
	100	28	5	17	28	33	22,2
	1000	35	40	8	38	38	31,8
	10000	230	321	286	332	282	290,2
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Merge Sort	10	30	26	28	26	29	27,8
	100	28	30	8	32	30	25,6
	1000	42	33	18	13	18	24,8
	10000	265	285	306	254	328	287,6
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Insertion Sort	10	32	4	29	26	29	24
	100	29	28	4	5	27	18,6
	1000	53	22	17	12	14	23,6
	10000	888	873	1141	938	1069	981,8
		1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Tempo Médio
Selection Sort	10	41	18	21	26	25	26,2
	100	30	28	23	3	3	17,4
	1000	52	19	28	16	29	28,8
	10000	1211	1423	1198	1203	1160	1239

Apêndice H – Teste do React Native realizado com lista pré-determinada no Android.

		1ª	2ª	3ª	4ª	5ª	Tempo
		Execução	Execução	Execução	Execução	Execução	Médio
Bubble Sort	10	146	19	20	10	14	41,8
	100	11	22	18	19	18	17,6
	1000	23	74	55	48	65	53
	10000	1951	1672	1810	1604	1573	1722
		1ª	2ª	3ª	4ª	5ª	Tempo
		Execução	Execução	Execução	Execução	Execução	Médio
Quick Sort	10	159	11	20	13	9	42,4
	100	14	15	13	11	13	13,2
	1000	168	31	32	32	47	62
	10000	492	507	445	492	379	463
		1ª	2ª	3ª	4ª	5ª	Tempo
		Execução	Execução	Execução	Execução	Execução	Médio
Merge Sort	10	162	13	10	5	8	39,6
	100	14	11	15	9	10	11,8
	1000	41	27	22	25	29	28,8
	10000	609	302	412	446	441	442
		1ª	2ª	3ª	4ª	5ª	Tempo
		Execução	Execução	Execução	Execução	Execução	Médio
Insertion Sort	10	162	11	9	16	14	42,4
	100	10	20	12	13	17	14,4
	1000	50	38	43	48	43	44,4
	10000	702	655	727	650	728	692,4
		1ª	2ª	3ª	4ª	5ª	Tempo
		Execução	Execução	Execução	Execução	Execução	Médio
Selection Sort	10	12	12	7	15	9	11
	100	12	11	10	13	16	12,4
	1000	45	37	44	37	36	39,8
	10000	1638	1351	1345	1537	1285	1431,2

