**UNIVERSITETI I PRISHTINËS “HASAN PRISHTINA”**

**FAKULTETI I INXHINIERISË ELEKTRIKE DHE KOMPJUTERIKE**



**PUNIM DIPLOME**

**Zhvillimi i aplikacionit për vizualizimin e algoritmeve të sortimit me Python/Pycharm**

**Mentori: Kandidati:**

Prof. Avni Rexhepi, Dr. Sc. Muhamed Hyseni

Prishtinë, Shtator 2024

**Abstrakti**

Përdorimi i algoritmeve të sortimit luan një rol kyç në funksionimin e aplikacioneve moderne duke ndikuar drejtpërdrejt në organizimin e të dhënave. Njohja e funksionimit të algoritmeve të sortimit është thelbësore për përzgjedhjen e algoritmit të duhur në mënyrë që puna të kryhet me efikasitet sa më të lart. Si inxhinierë të kompjuterikës, ne duhet të jemi të aftë të vlerësojmë se cilat algoritme t’i zgjedhim varësisht nga kërkesat e aplikacioneve në të cilat punojmë.

Në këtë punim është përfshirë krijimi i një desktop aplikacioni për vizualizimin e algoritmeve të sortimit si: Merge Sort, Counting Sort dhe Bucket Sort. Aplikacioni është zhvilluar në gjuhën programuese Python duke përdorur veglën për mundësimin e mjedisit të zhvillimit PyCharm dhe libraritë përkatëse të Python-it. Qëllimi kryesor i këtij punimi është analizimi i secilit prej algoritmeve, vizualizimi i tyre dhe krahasimi i performancave të tyre. Gjithashtu është bërë analiza e kompleksitetit kohor të këtyre algoritmeve dhe kemi shqyrtuar rastin më të mirë, mesatar dhe më të keq të tyre. Varësisht nga këto veti kemi shqyrtuar se ku mund të përdoren këto algoritme në mënyrë optimale.

**Abstract**

The use of sorting algorithms plays a key role in the functioning of modern applications, directly affecting the organization of data. Understanding the workings of sorting algorithms is essential for selecting the appropriate algorithm so that the work is performed as efficiently as possible. As computer engineers, we must be capable of evaluating algorithms based on the requirements of the applications we are working on.

This paper includes the creation of a desktop application for visualizing sorting algorithms such as Merge Sort, Counting Sort, and Bucket Sort. The application is developed in the Python programming language using the PyCharm development environment and the relevant Python libraries. The main goal of this paper is to analyze each of the algorithms, visualize them, and compare their performances. Additionally, we have analyzed the time complexity of these algorithms and examined their best, average, and worst cases. Based on these properties, we have reviewed where these algorithms can be used optimally.

.

**Përmbajtja**

[Hyrje 5](#_Toc176128258)

[1 Algoritmet 6](#_Toc176128259)

[1.1 Çka janë algoritmet 6](#_Toc176128260)

[1.2 Analiza e Algoritmeve 7](#_Toc176128261)

[1.3 Përdorimi i Algoritmeve 9](#_Toc176128262)

[2 Algoritmet e Sortimit 10](#_Toc176128263)

[2.1 Çfarë janë algoritmet e sortimit? 10](#_Toc176128264)

[2.2 Vetitë dhe Përdorimi 11](#_Toc176128265)

[2.3 Algoritmi Merge Sort 12](#_Toc176128266)

[2.4 Algoritmi Counting Sort 14](#_Toc176128267)

[2.5 Algoritmi Bucket Sort 17](#_Toc176128268)

[2.6 Merge Sort vs Counting Sort vs Bucket Sort 19](#_Toc176128269)

[3 Teknologjitë e përdorura 20](#_Toc176128270)

[3.1 Procesi për zgjedhjen e teknologjisë 20](#_Toc176128271)

[3.2 Python 20](#_Toc176128272)

[3.3 PyCharm 21](#_Toc176128273)

[4 Aplikacioni 22](#_Toc176128274)

[4.1 Interface i Aplikacionit 22](#_Toc176128275)

[4.2 Merge Sort 27](#_Toc176128276)

[4.3 Counting Sort 31](#_Toc176128277)

[4.4 Bucket Sort 34](#_Toc176128278)

[Përfundimet 37](#_Toc176128279)

[Summary 38](#_Toc176128280)

[Shtesat 39](#_Toc176128281)

[Referencat 40](#_Toc176128282)

# 

# Hyrje

Ky punim diplome ka për qëllim analizën dhe shtjellimin e algoritmeve Merge sort, Counting sort dhe Bucket sort, gjithashtu si detyrë na mbetet vizualizimi i tyre.

Algoritmet luajnë rol shumë të rëndësishëm për shpejtësinë dhe efikasitetin e programeve në përgjithësi. Aplikacionet të cilat i përdorim, në prapavijë kanë një algoritëm i cili merret me renditjen e të dhënave, manipulimin me to etj. Për ta përzgjedhur algoritmin të cilin do të përdoret, programeri duhet të marrë parasysh shumë faktorë dhe variabla varësisht punës së cilës do ta kryej aplikacioni, resurset, llojin e veglave të cilat i përdorë etj. Për ta zgjedhur algoritmin fillimisht programeri duhet ta kuptoj se si punon algoritmi dhe më pas të marrë parasysh faktorët e tjerë.

Një nga qëllimet kryesore të këtij punimi është krijimi i një desktop aplikacioni i cili bën vizualizimin e algoritmeve: Merge sort, Counting sort dhe Bucket sort. Me vizualizim kuptohet shfaqja në ekran e ekzekutimit të algoritmit gjatë sortimit të një vargu dhe sqarimi i disa hapave kyç gjatë veprimeve të kryera. Sfida kryesore e këtij punimi ka qenë krijimi i aplikacionit me një pamje miqësore për përdoruesit duke përdorur gjuhën programuese python.

# Algoritmet

## Çka janë algoritmet

“Në Shkenca Kompjuterike, një algoritëm është një listë e instruksioneve të cilat përdoren për të zgjidhur probleme ose për të kryer punë, duke u bazuar në kuptimin e alternativave në dispozicion. Algoritmet janë më shumë se programim, ato janë specifikime për performimin e kalkulimeve, procesimit të të dhënave, për logjikë automatike ose vendimmarrje.” (1)

Një algoritëm është procedurë për zgjidhjen e një problemi të definuar kompjuterik. Zhvillimi dhe analiza e algoritmeve është fundamentale për të gjitha aspektet në shkenca kompjuterike, si në: ineteligjencë artificiale, baza të të dhënave, rrjeta, sisteme operative, sigurisë dhe të tjera. Për të zhvilluar algoritme kërkohet një kuptueshmëri e alternativave në dispozicion për zgjidhjen e problemeve kompjuterike, duke përfshirë hardware-in, rrjetat, gjuhën programuese dhe pengesat e performancës të cilat i shtohen çdo zgjidhje të caktuar. (2)

Në bazë të definiconeve të dhëna më sipër, algoritmit mund t’i referohemi si një procedurë për zgjidhjen e një problemi, nëse planifikohet që të zhvillohet ose përdoret një algoritëm i gatshëm, parasysh duhet të kemi shumë faktorë si: gjuha programuese e cila do të përdoret, burimet në dispozicion, siguria e të tjera. Ka standarde të ndryshme për ndarjen e algoritmeve, një algoritëm mund të bëj pjesë në grupe të ndryshme varësisht prej parametrit në bazë të të cilit është bërë ndarja.

Ndarja në bazë të strukturës dhe llojit të zgjidhjes së poblemit: (3)

1. **Algoritmi i Programimit Dinamik (Dynamic Programming Algorithm)** – Ky algoritëm funksionon duke i mbajtur në mend rezultatet e një ekzekutimi paraprak dhe duke i përdorur ato për të arritur rezultate të reja.
2. **Algoritmi i Forcës Brute (Brute Force Algorithm)** – Ky algoritëm përfshinë provimin e të gjitha zgjidhjeve të mundshme për të arritur zgjidhjen e saktë, por merr shumë kohë.
3. **Algoritmi Rekursiv (Recursive Algorithm)** – Ky algoritëm përfshinë thirrjen e vetvetes me vlera më të vogla deri sa të zgjidhë problemin.
4. **Algoritmi Lakmitar (Greedy Algorithm)** – Ky algoritëm merr zgjidhjen më të mirë lokale duke shpresuar që është zgjidhja më e mirë globale në fund.
5. **Algoritmi Ndaj dhe Sundo (Divide and Conquer Algorithm)** – Ky algoritëm e bën ndarjen e problemit në nënprobleme, i zgjidhë dhe i bashkon zgjidhjet.
6. **Algoritmi i kthimit prapa (Backtracking Algorithm)** – Ky algoritëm përfshinë gjetjen e zgjidhjeve të ndryshme dhe në fund kthehet ta gjejë zgjidhjen e duhur pasi të mos ketë zgjidhje të mundshme.
7. **Algoritmet e rastësishme (Randomized algorithms)** – Algoritmet e rastësishme përdorin numra rastësorë për të marrë vendime gjatë punës, që do të thotë se mund të kenë përfundime të ndryshme sa herë të ekzekutohen.

## Analiza e Algoritmeve

Analiza e algoritmeve na e mundëson llogaritjen e kohës për zgjidhjen e një problemi nga një algoritëm. Duke ditur kohëzgjatjen e algoritmeve për zgjidhjen e një problemi me N vlera hyrëse, mund të përcaktohet se cili algoritëm është më i përshtatshëm për zgjidhjen e problemit tonë. Por ka raste kur kemi disa algoritme të cilat e zgjidhin problemin për kohën e njëjtë, cilit aspekt i kushtohet rëndësi në këtë rast? Zakonisht me analizën e algoritmeve prioritet i jepet kohëzgjatjes së zgjidhjes së problemit, por ka raste kur hapësira mund të jetë aspekt i rëndësishëm i cili po ashtu duhet të merret parasysh. Arsyeja pse të dy këto vlera janë të rëndësishme është llogaritja e efikasitetit të algoritmeve për probleme të ndryshme. Analiza e një algoritmi nuk matet vetëm me kohën e shprehur në sekonda ose numrin e cikleve kompjuterike për shkak se atëherë, kompjuterët me kapacitet më të madh do ta zgjidhnin problemin më shpejt sesa kompjuterët me kapacitet më të ulët, por qëllimi kryesor është krahasimi ndërmjet algoritmeve të llojit të njëjtë në mes vete.

Gjatë analizës së algoritmeve merren parasysh këta parametra:

1. **Klasat hyrëse –** Hyrja është shumë e rëndësishme për analizën e algoritmeve sepse përcakton shtegun e ekzekutimit të një algoritmi p.sh. një varg rritës mund të ekzekutohet me hapa të ndryshëm nga një varg zbritës.
2. **Kompleksiteti hapësinor** – Ky parametër ka të bëj me hapësirën që i duhet algoritmit për të funksionuar. Ka qenë shumë i rëndësishëm në të kaluarën për shkak të limitimeve të resurseve të kompjuterit.
3. **Rastet për shqyrtim**:

* **Rasti më i keq(Worst Case)** – Përfshinë rastin i cili merr kohën më të gjatë për t’u zgjidhur, me fjalë të tjera përfshinë hyrjen më të keqe për zgjidhjen e problemit. Tek algoritmet e kërkimit është rasti kur elementi gjendet në fund të vargut ose nuk gjendet fare në varg.
* **Rasti më i mirë(Best Case) –** Përfshinë rastin i cili merr kohën më të shkurtër për t’u zgjidhur, me fjalë të tjera përfshinë kushtet optimale për zgjidhjen e problemit. Tek algoritmet e kërkimit është rasti kur elementi gjendet në krye të vargut.
* **Rasti mesatar(Average Case) –** Përdoret për të shpjeguar sjelljen e një algoritmi në kushte mesatare, pra efikasitetin mesatar të një algoritmi. Kjo analizë është më e vështira për t’u bërë sepse përfshinë shumë detaje. Hapi i parë është ndarja në grupe e seteve hyrëse. Hapi i dytë ka të bëjë me përcaktimin e probabilitetit se prej cilit grup do të vijë hyrja. Hapi i tretë përcakton se sa gjatë do të ekzekutohet algoritmi për secilën prej këtyre grupeve. Duhet të kemi parasysh se secila hyrje, në secilin grup duhet të marr kohën e njëjtë, nëse ky kusht nuk plotësohet atëherë grupin në fjalë duhet t’a ndajmë në dy grupe të ndara e më pas kohën mesatare e llogarisim me formulën e mëposhtme:

**A(n) =**

Ku **n** është madhësia e hyrjes, **m** është numri i grupeve, është gjasa që hyrja do të jetë nga grupi **i**, dhe është koha që e merr algoritmi për një hyrje nga grupi **i**.

Në disa raste gjatë kësaj analize konsiderohet se të gjitha grupet hyrëse kanë probabilitet të barabartë, dhe rastin mesatar mund t’a llogarisim me formulën e mësipërme apo duke përdorur një formulë të thjeshtuar, si në vijim:

**A(n) =**

* **Shpejtësitë/Shkallët e rritjes** – paraqet shpejtësinë/shkallën e rritjes së operacioneve të një algoritmi për zgjidhjen e një problemi teksa rritet madhësia e problemit. Gjatë analizës së algoritmit, rëndësi më të madhe kanë rastet me sete më të mëdha të dhënash sesa e kundërta, andaj edhe klasa e rritjes është shumë e rëndësishme. Pasi që shkalla/shpejtësia e rritjes së një algoritmi është e rëndësishme dhe pasi që shkalla/shpejtësia e rritjes është e dominuar nga termi më i madh në një ekuacion, termat që rriten më ngadalë do të hidhen poshtë. Kur largohen të gjitha këto aspekte, mbesin me rendin e funksionit ose algoritmit të ndërlidhur. Pastaj algoritmet mund të grupohen së bashku në bazë të rendit të tyre. Ato ndahen në tri kategori – ato që rriten së paku aq shpejt sa disa funksione, ato që rriten me shkallë/shpejtësi të njëjtë dhe ato që rriten më shpejt.
* **Big Omega** - paraqet klasën e funksioneve që rriten së paku aq shpejt sa funksioni f, mund t’a kuptojmë si përcaktim të kufirit të poshtëm të funksionit. Formalisht, kjo do të thotë se nëse **g(x) ∈ ,**

**g(n) ≥ c f(n)** për çdo **n ≥** ( ku c është konstantë pozitive).

* **Big Oh** – **О(f)** paraqet klasën e funksioneve të cilat rriten jo më shpejt sesa f, **О(f)** mund t’a kuptojmë si kufi të epërm të funksionit. Formalisht kjo do të thotë se nëse **g(x) ∈ О(f), g(n) ≤ c f(n)** për çdo **n ≥** .
* **Big Theta** - **ϴ(f)** paraqet klasën e funksioneve të cilat rriten me shpejtësi të njëjtë sikurse funksioni f. Formalisht **ϴ(f)** shihet si vendi ku big omega dhe big oh takojnë njëra tjetrën(përputhen) kështu **ϴ(f) = ⋂ О(f).(4)**

## Përdorimi i Algoritmeve

Algoritmet përdoren shumë në përditshmëri, shpesh herë edhe pa vetëdijen e përdoruesit. Sa herë që kërkohet diçka nëpër aplikacione, rekomandimet që ofrohen janë frut i një algoritmi. Sa herë që ekzekutohet një program në prapaskenë, është duke u përdorur një algoritëm. Algoritmet janë një set i instruksioneve të cilat kryejnë një punë të caktuar, çka i bën ato bazën e informatikës. Algoritmet përdoren në të gjitha fushat e shkencave komjupterike.

Në bazë të llojit të përdorimit, algoritmet ndahen në:

1. **Algoritme të kërkimit (Search Algorithms)** – Algoritme të dizajnuara për të gjetur dhe kthyer informacione në strukturën e të dhënave p.sh. algoritmi për kërkim binar.
2. **Algoritme të sortimit (Sorting Algorithms)** – Algoritme të cilat e renditin një varg sipas një rregulli të caktuar, p.sh Merge sort.
3. **Algoritmet për grafe (Graph algorithms)** – Algoritme të cilat punojnë me grafe, qëllimi i tyre është për të gjetur rrugën më të shkurtër në mes nyjeve , p.sh algoritmi Dijkstra.
4. **Algoritmet për enkriptim (Encypting algorithms)** – Algoritme të cilat duke kryer operacione matematikore, transformojnë tekstin e thjesht në tekst cypher, p.sh AES, Blowfish, Camellia etj.
5. **Algoritmet për kompresim (compressing algorithms)** – Algoritmet të cilat duke larguar informata të panevojshme nga të dhënat, zvogëlojnë madhësinë e tyre, p.sh Hufmann encoding, PPM, RLE etj.
6. **Algoritmet për machine learning (machine learning algorithms)** – Algoritme të cilat në përmbajtjen e tyre kanë një set procesesh të përdorura nga AI për të kryer detyra, p.sh Linear regression, Mean, Decision Tree etj.

# Algoritmet e Sortimit

## Çfarë janë algoritmet e sortimit?

Sortimi është koncept i thjesht i cili përfshinë radhitjen e elementeve të një vargu sipas një modeli të caktuar rritës ose zvogëlues. Një model rritës i rendit elementet duke filluar nga më i vogli deri tek elementi më i madh, kurse modeli zvogëlues elementin e parë e ka elementin me vlerën më të madhe dhe elementin e fundit me vlerën më të vogël. Algoritmet e sortimit janë një set rregullash karakteristike të cilat kryejnë sortimin.

Disa prej algoritmeve më të njohura të sortimit janë: Insertion Sort, Bubble Sort, Shell Sort, Radix Sort, Merge Sort etj.

Algoritmet e Sortimit ndahen në:

1. Algoritme të sortimit të bazuara në krahasim - janë algoritme tek të cilat përdoren operatorët “më i vogël ose i barabartë” për të vendosur se cili element vendoset para tjetrit gjatë sortimit.
2. Algoritme të sortimit jo të bazuara në krahasim – janë algoritme të cilat nuk i krahasojnë elementet e vargut ndërmjet vete për t’i krahasuar.

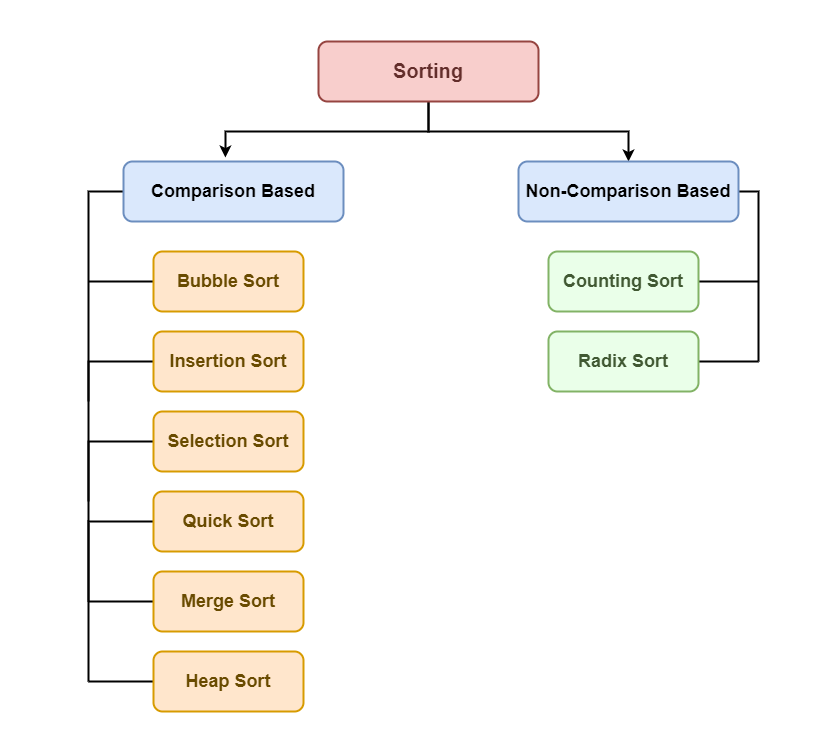


Figura 1 - Algoritmet krahasuese dhe algoritmet jo krahasuese

Algoritmet e sortimit të cilat do të trajtohen në këtë punim janë dy të bazuara në krahasim(Merge sort dhe Bucket sort) dhe një jo krahasues(Counting sort).

## Vetitë dhe përdorimi

Disa prej vetive kryesore të algoritmeve të sortimit:

* **Kompleksiteti kohor:** Një prej vetive kryesore të algoritmeve të sortimit është kompleksiteti kohor, i cili përcakton kohën e nevojshme për të kryer sortimin në varësi të numrit të elementeve. Algoritmet klasike si Bubble Sort dhe Insertion Sort kanë kompleksitet O(n²), ndërsa algoritme më të avancuara si Merge Sort dhe Quick Sort kanë kompleksitet O(n log n).
* **Kompleksiteti hapësinor:** Ky tregon hapësirën e kujtesës së nevojshme për të ekzekutuar algoritmin. Algoritmet që kërkojnë hapësirë shtesë, si Merge Sort, kanë kompleksitet hapësinor më të lartë krahasuar me ato që janë in-place, si Quick Sort.
* **Stabiliteti:** Një algoritëm sortimi është stabil nëse ruan rendin relativ të elementeve me vlera të njëjta. Kjo është e rëndësishme në aplikime ku rendi fillestar i të dhënave ka kuptim të veçantë. Merge Sort dhe Bubble Sort janë shembuj të algoritmeve stabile.
* **Krahasimi dhe Jo-Krahasimi:** Algoritmet e sortimit mund të ndahen në ato që bazohen në krahasimin e elementeve (si Quick Sort) dhe ato që nuk bazohen në krahasim (si Radix Sort). Algoritmet e krahasimit kanë një kufi më të ulët teorik O(n log n), ndërsa ato të jo-krahasimit mund të arrijnë efikasitet më të lartë për raste specifike.

Përdorimi i algoritmeve të sortimit është shumë i rëndësishëm sidomos në përpunimin e të dhënave në ambiente me numër të madh të të dhënave. Në problemet e optimizimit, si planifikimi i detyrave, algoritmet e sortimit luajnë një rol të rëndësishëm në radhitjen e prioriteteve ose optimizimin e burimeve në mënyrë që të përmirësohet efikasiteti i sistemit.

## Algoritmi Merge Sort

Merge Sort është një nga algoritmet më të vjetra dhe më me impakt të renditjes, i shpikur nga John Von Neumann në vitin 1945. Ky algoritëm u zhvillua gjatë periudhës kur Von Neumann po punonte në një nga kompjuterët e parë elektronik të përgjithshëm, EDVAC (Electronic Discrete Variable Automatic Computer).

Merge sort është një algoritëm “ndajë dhe sundo” (divide and conquer) që ka një kompleksitet të qëndrueshëm O(nlog n), ku n është numri i elementeve që duhen renditur. Stabiliteti i tij e bën të rëndësishëm Merge sortin për sortimin e grupeve të mëdha me të dhëna si dhe menaxhimin e të dhënave. Merge sort funksionon në dy faza kryesore: Ndarja dhe Bashkimi. Gjatë ndarjes rreshti fillestar ndahet në dy grupe të barabarta nëse është e mundur ose pothuajse të barabarta, kjo ndarje vazhdon derisa nëngrupet mbesin vetëm me nga një element. Kur përfundon ndarja fillon bashkimi i grupeve si dhe renditja sipas rregullës.

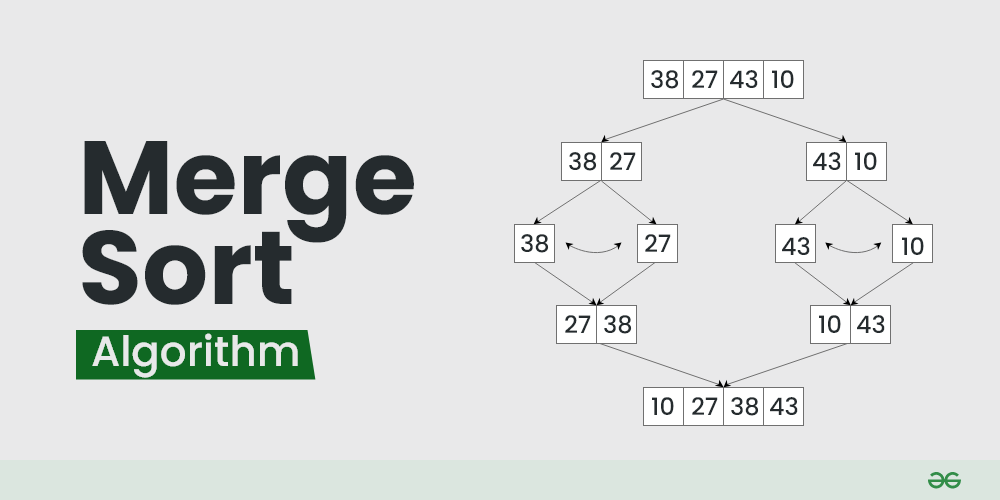


Figura 2 - Ilustrim i Merge Sort

Tek Merge Sort rasti më i mirë, rasti mesatar, dhe rasti më i keq kanë kompleksitet kohor të njëjtë.

* **Rasti më i mirë**

**Kompleksiteti kohor**: O(nlogn)

**Përshkrimi**: Edhe në rastin kur elementet në varg janë të renditura, Merge Sort gjithmonë i ndan grupet dhe nëngupet dhe kryen procesin e bashkimit për çdo ndarje. Kjo do të thotë që në rastin më të mirë (p.sh., të dhënat janë tashmë të renditura), algoritmi prap do t’i kryej të gjithë hapat e paraparë.

* **Rasti mesatar**

**Kompleksiteti kohor**: O(nlogn)

**Përshkrimi**: Në mesataren e të gjitha rasteve të mundshme të renditjes së të dhënave, Merge Sort kryen ndarjen dhe bashkimin në mënyrë të njëjtë, duke çuar në kompleksitetin O(nlogn). Ky është një avantazh sepse algoritmi është efikas në mënyrë konstante, pavarësisht nga renditja fillestare e të dhënave.

* **Rasti më i keq**

**Kompleksiteti kohor**: O(nlogn)

**Përshkrimi**: Edhe në rastin më të keq (p.sh., kur të dhënat janë të renditura në mënyrë të kundërt), Merge Sort gjithmonë ndan grupet në dy dhe bashkon nën-grupet me një kompleksitet që mbetet O(nlogn). Në këtë aspekt, ai është më i qëndrueshëm se disa algoritme të tjerë, si Quick Sort, i cili ka një kompleksitet O(n²) në rastin më të keq.

## Algoritmi Counting Sort

Algoritmi Counting sort është një algoritëm i sortimit jo-krahasues që përdoret më së mirë kur vlerat e të dhënave janë të kufizuara në një diapazon të caktuar. Ky algoritëm është veçanërisht efikas kur kemi një numër të madh të elementeve me një gamë të vogël vlerash.

Hapat e ekzekutimit të Counting sort janë:

1. **Numërimi i frekuencave:**
   * Krijohet një varg (i quajtur count) që ruan numrin e herëve që çdo element shfaqet në vargun origjinal.
   * Madhësia e vargut count është sa vlera maksimale e elementit në vargun origjinal plus një.
2. **Akumulimi:**
   * Array count përditësohet për të mbajtur pozicionin e fundit të çdo elementi në vargun e renditur.
   * Duke iteruar mbi vargun count, çdo vlerë në këtë array përditësohet duke shtuar vlerën e mëparshme.
3. **Rendisja e elementeve:**
   * Një varg i ri (i quajtur output) përdoret për të vendosur elementet në pozicionet e tyre të renditura.
   * Duke kaluar nga fundi në fillim të vargut origjinal, për çdo element, gjejmë pozicionin e tij në vargun output duke përdorur vargun count, dhe e zvogëlojmë vlerën përkatëse në count.
4. **Kopjimi i Rezultatit:**
   * Elementet e renditura në output kopjohen mbrapsht në vargun origjinal.

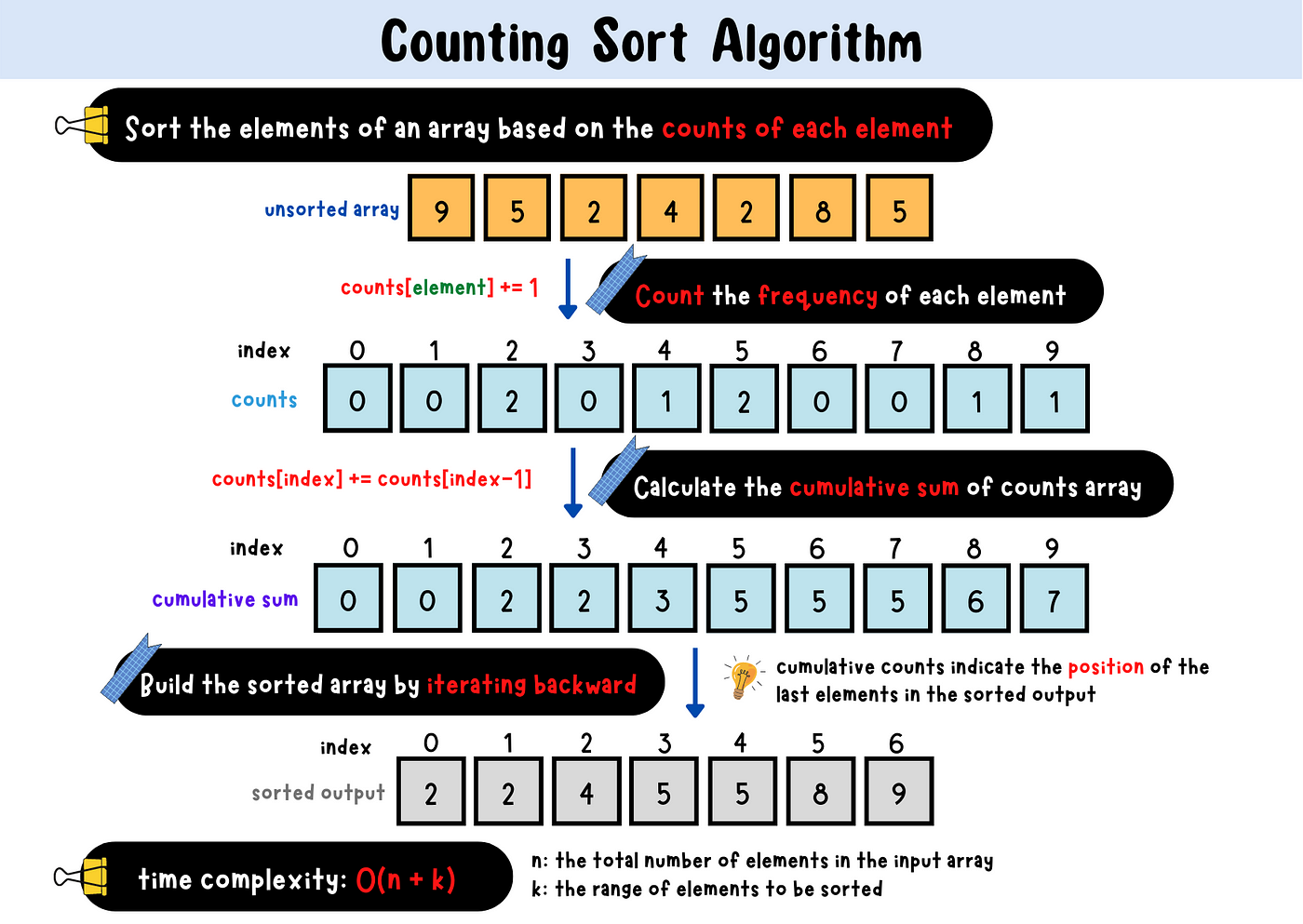
****

Figura 3 - Ilustrim i Couting Sort

Counting sort ka një kompleksitet kohor O(n+k) për të gjitha rastet.Ku “n” përfaqëson numrin e elementeve në vargun që duhet të renditen dhe “k” përfaqëson gamën e mundshme të vlerave që elementet në array mund të marrin. Counting sort ka kompleksitet të njëjtë në të gjitha rastet për shkak se Counting sort nuk është një algoritëm krahasues, dhe nuk ndikohet nga mënyra se si janë të renditura ose të shpërndara të dhënat.

* **Rasti më i mirë:**

**Kompleksiteti kohor**: O(n+k)

**Përshkrimi**: Në rastin më të mirë, vargu mund të jetë i renditur që në fillim, por Counting sort do të kryejë të njëjtat hapa si gjithmonë, kështu që koha e ekzekutimit mbetet O(n+k).

* **Rasti mesatar:**

**Kompleksiteti kohor**: O(n+k)

**Përshkrimi**: Në rastin mesatar, kur elementet janë të shpërndarë në mënyrë të rastësishme, algoritmi përsëri do të kalojë nëpër të gjithë vargun dhe do të ndërtojë vargun count, që kërkon O(n+k) kohë.

* **Rasti më i keq:**

**Kompleksiteti kohor**: O(n+k)

**Përshkrimi**: Në rastin më të keq, kur elementet janë të renditura në rend të kundërt ose kanë shpërndarje jo të barabartë, counting sort përsëri do të kryejë të njëjtat hapa me të njëjtin kompleksitet kohor O(n+k).

# 

## Algoritmi Bucket Sort

Bucket Sort është një algoritëm renditjeje që funksionon duke ndarë të dhënat në disa grupe më të vogla (të quajtura "kovë" ose "bucket"), dhe pastaj rendit secilën kovë individualisht. Pasi të gjitha kovat të jenë renditur, ato bashkohen për të krijuar një grup të vetëm të renditur.

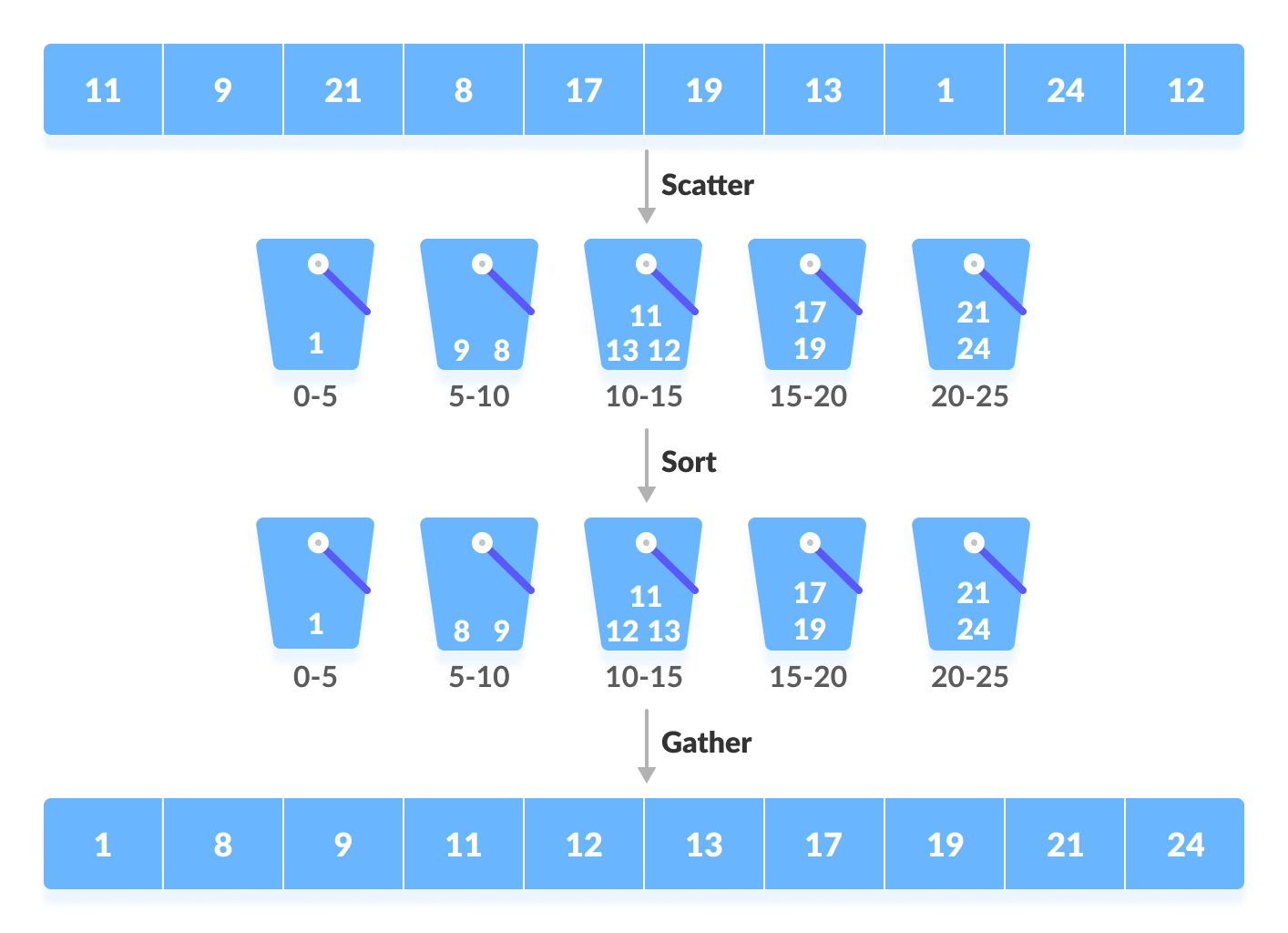


Figura 4 - Ilustrim i Bucket Sort

Bucket sort është një algoritëm renditjeje stabil. Bucket sort është veçanërisht i efektshëm kur elementet e të dhënave kanë një shpërndarje uniforme ose të njohur, si dhe kur kemi një numër të madh të kovëve në mënyrë që elementet të ndahen në mënyrë të barabartë. Ky algoritëm është përdorur shpesh në aplikacione që përfshijnë renditjen e numrave me presje të lëvizshme ose të dhëna të tjera që mund të ndahen lehtësisht në intervale.

* **Rasti më i mirë**

**Kompleksiteti kohor**: O(n+k)

**Përshkrimi**: Rasti më i mirë ndodh kur elementet janë shpërndarë në mënyrë të barabartë nëpër kova. Në këtë rast, secila kovë përmban një numër të vogël elementesh, dhe renditja e secilës kovë (që zakonisht bëhet me një algoritëm të thjeshtë si Insertion Sort) është shumë e shpejtë. Kompleksiteti për këtë rast përfshin ndarjen e elementeve në kovë (O(n)) dhe renditjen e kovëve (O(k)), ku “k” është numri i kovave dhe “n” numri i elementeve të cilat duhen renditur.

* **Rasti mesatar**

**Kompleksiteti kohor**: O(n+k)

**Përshkrimi**: Në shumicën e rasteve, Bucket Sort punon mirë kur të dhënat janë shpërndarë në mënyrë të moderuar nëpër kova. Rasti mesatar është kur kovat përmbajnë një numër të ngjashëm elementesh, duke çuar në kompleksitet O(n+k). Ky kompleksitet përfshin kohën e ndarjes së elementeve në kovë dhe kohën për renditjen e secilës kovë.

* **Rasti më i keq**

**Kompleksiteti kohor**: O()

**Përshkrimi**: Rasti më i keq ndodh kur të gjithë elementet përfundojnë në të njëjtën kovë. Në këtë rast, Bucket Sort degradon në një algoritëm renditjeje të thjeshtë brenda kësaj kove, si Insertion Sort, që ka kompleksitet O(). Ky skenar ndodh kur të dhënat nuk janë shpërndarë mirë dhe kovët janë shumë të pabalancuara.

## Merge Sort vs Counting Sort vs Bucket Sort

Merge Sort ndërtohet në mënyrë rekursive duke ndarë dhe kombinuar (Divide and Conquer). Ky algoritëm ka kompleksitet kohor konstant gjatë të gjitha rasteve andaj njihet si algoritëm stabil. Merge Sort përdoret për një gamë të gjerë të dhënash madje edhe në rastet kur kemi të bëjmë me databaza gjigande. Prej disavantazheve të tij, është se ai kërkon hapësirë shtesë O(n) për të bashkuar nënlistat dhe është relativisht i ndërlikuar për implementim. Counting Sort në anën tjetër është shumë efikas në vargje me gamë të vogël të elementeve dhe është stabil nëse implementohet me kujdes. Ai përshtatet shumë mirë për të dhëna të tëra pozitive dhe me rang të kufizuar. Disavantazhet e tij janë se nuk është i përshtatshëm për të dhëna të shpërndara në një gamë të gjerë, nuk punon drejtpërdrejt për lloje të tjera të dhënash përveç numrave të plotë (integer). Kurse Bucket Sort është efikas për të dhëna që shpërndahen njëtrajtësisht në një gamë të caktuar, ka shpejtësi më të lartë sesa algoritmet krahasuese për të dhëna me shpërndarje të mire dhe është shumë i përshtatshëm për të dhëna me formate që janë më shumë se numra të plotë.

Në jetën e përditshme Merge Sort do të përdorej për **renditjen e të dhënave financiare në një databazë të madhe. Couting Sort do ishte i përshtatshëm për** renditjen e notave të studentëve nga 0 deri në 100. Bucket Sort do të mund të përdorej **për renditjen e rezultateve të performances së një punimi shkencor ku këto rezultate do ishin numra real (midis 0 dhe 1).**

# Teknologjitë e përdorura

## Procesi për zgjedhjen e teknologjisë

Procesi i zgjedhjes së teknologjisë për krijimin e aplikacionit mund të konsumojë kohë për një programer për shkak të opsioneve të shumta si dhe kritereve të preferueshme në lidhje me funksionet e aplikacionit. Për aplikacionin për vizualizim të algoritmeve janë marrë parasysh disa faktorë si: kërkesat të cilat duhen plotësuar, përshtatshmëria e gjuhës programuese dhe vegla e cila është vendosur të përdoret dhe të tjera. Duke marrur parasysh të gjithë këta faktor, ky punim është realizuar duke përdorur gjuhën programuese python dhe veglën PyCharm. Gjuha programuese python ofron një numër të madh bibliotekash dhe modulesh për funksionalitetin e aplikacionit, prej tyre janë përdorur: tkinter, matplotlib, numpy dhe PIL. Python ka një mbështetje të shkëlqyer për algoritmet dhe analizën e të dhënave përmes bibliotekave si numpy.

## Python

Python është një gjuhë programimi e nivelit të lartë, e njohur për lehtësinë e përdorimit dhe qartësinë e saj. Ajo përdoret gjerësisht për zhvillimin e aplikacioneve të ndryshme, duke përfshirë ueb, bazat e të dhënave, automatizimin, inteligjencën artificiale dhe më shumë. Ja disa karakteristika kryesore të Python-it:

1. **Sintaksë e thjeshtë dhe e lexueshme:** Python-i është i njohur për sintaksën e tij që i ngjan shumë gjuhës së përditshme, gjë që e bën atë të lehtë për t'u mësuar dhe kuptuar.
2. **Interpretues:** Python është një gjuhë interpretuese, që do të thotë se kodet ekzekutohen rresht për rresht, pa pasur nevojë për një fazë të veçantë të ndërtimit të kodit.
3. **Tipizim dinamik:** Python-i përdor tipizimin dinamik, që do të thotë se nuk ka nevojë të deklarosh llojet e të dhënave; ato caktohen automatikisht bazuar në përmbajtjen e ndryshoreve.
4. **Mbështetje për stilet e ndryshme të programimit:** Python-i mbështet programimin objekt-orientuar, procedural dhe funksional, duke e bërë atë shumë fleksibil për zbatime të ndryshme.
5. **Biblioteka të fuqishme:** Python-i ka një ekosistem të gjerë bibliotekash dhe paketash që lejojnë zhvilluesit të kryejnë detyra të ndryshme pa pasur nevojë të shkruajnë kod nga fillimi.
6. **Multiplatformë:** Python-i funksionon në platforma të ndryshme si Windows, macOS, dhe Linux pa kërkuar ndryshime në kod.

## PyCharm

PyCharm është një Mjedis Zhvillimi të Integruar (IDE) për gjuhën Python, i zhvilluar nga JetBrains. Ai është një nga IDE-të më të njohura për zhvilluesit e Python-it dhe ofron një sërë karakteristikash që e bëjnë zhvillimin më të lehtë dhe më efikas. Ja disa nga tiparet kryesore të PyCharm:

1. **Redaktori i avancuar i kodit:** PyCharm ofron një redaktues të fuqishëm me funksionalitete si: ndriçimi i sintaksës, auto-kompletimi i kodit, dhe sugjerimet inteligjente. Ai gjithashtu mbështet refaktorizimin e kodit për të përmirësuar strukturën e tij pa e ndryshuar funksionalitetin.
2. **Debuggeri i integruar:** PyCharm ka një debugger të fuqishëm që lejon zhvilluesit të ekzekutojnë kodin e tyre hap pas hapi, të vendosin pika ndalimi (breakpoints), dhe të inspektojnë vlerat e ndryshoreve për të identifikuar dhe rregulluar gabimet në kod.
3. **Integrimi me sistemet e kontrollit të versioneve:** PyCharm mbështet sisteme të ndryshme të kontrollit të versioneve, si Git, Mercurial, dhe SVN. Kjo e bën të lehtë menaxhimin e ndryshimeve në kod dhe bashkëpunimin me zhvillues të tjerë.
4. **Mbështetja për mjedise virtuale:** PyCharm lehtëson menaxhimin e mjediseve virtuale Python, duke bërë të mundur krijimin dhe përdorimin e mjediseve të veçanta për projekte të ndryshme, me biblioteka dhe varësi të ndara.
5. **Testimi i integruar:** PyCharm mbështet mjetet e testimit të Python si pytest dhe unittest, duke bërë të lehtë krijimin dhe ekzekutimin e testeve për kodin tuaj.
6. **Redaktimi i kodeve të tjera:** Përveç Python-it, PyCharm mbështet edhe gjuhë të tjera programimi dhe teknologji, si: JavaScript, HTML, CSS, dhe SQL, që e bën atë të përshtatshëm për zhvillimin e aplikacioneve ueb dhe të dhënave.
7. **Analiza statike e kodit:** PyCharm ofron mjete për analizimin e kodit për të identifikuar gabime të mundshme, probleme të stilit, dhe përmirësime të mundshme në performancë.
8. **Integrimi me Docker dhe shërbimet cloud:** PyCharm mund të integrohet me Docker për të menaxhuar konteinerët dhe me shërbime cloud për të zhvilluar dhe vendosur aplikacionet në re.
9. **Versione të ndryshme:** PyCharm vjen në dy versione:
   * **PyCharm Community Edition** - falas dhe me burim të hapur, që ofron të gjitha tiparet bazë për zhvillimin e Python-it.
   * **PyCharm Professional Edition** - me pagesë, që përfshinë të gjitha tiparet e Community Edition dhe shton mbështetje për teknologji të avancuara si zhvillimi i aplikacioneve ueb, analiza e të dhënave, dhe më shumë.

# Aplikacioni

## Interface i Aplikacionit

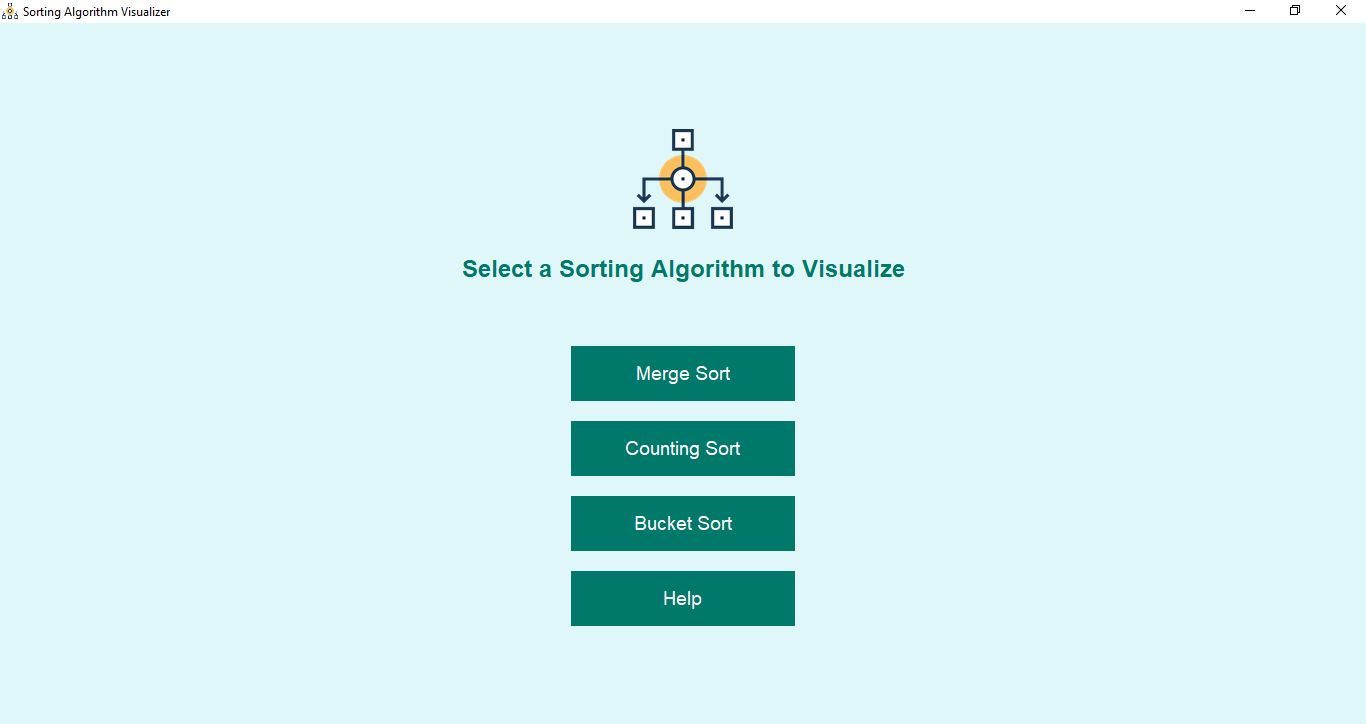


Figura 5 - Faqja kryesore e aplikacionit

Pamja fillestare e aplikacionit perfshinë katër butona me emrat: Merge Sort, Counting Sort, Bucket Sort dhe Help. Në këtë faqe përdoruesi përzgjedh llojin e algoritmit të cilin do ta vizualizoj dhe me anë të butonit “Help” përdoruesi mund t’i marr disa instruksione se si funksionon aplikacioni.

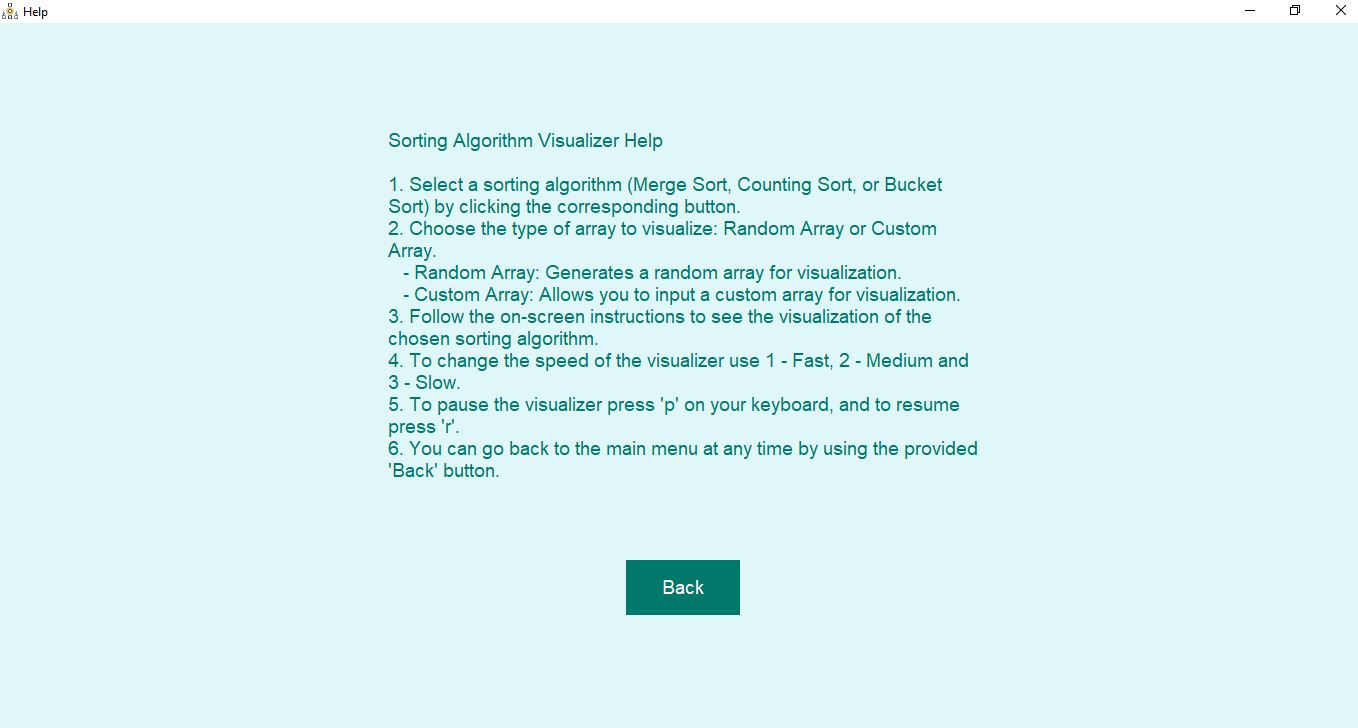


Figura 6 - Faqja e ndihmës

Kur klikojm butonin “Help” na hapet dritarja e ndihmës e cila na tregon se si ta përdorim aplikacionin. Fillimisht përdoruesi e zgjedh llojin e algoritmit sipas dëshirës duke klikuar në butonin korrespondues. Më pas, ai zgjedh a do të shkruaj të dhënat manualisht apo të gjenerohet një varg i rastësishëm. Vargu i rastësishëm gjenerohet vet, kurse për vargun manual, duhet të zgjedhet gjatësia e tij dhe më pas të futen të dhënat. Gjatë vizualizimit në ekran shfaqen shpjegime të procedurave që kryhen gjatë vizualizimit. Nëse përdoruesi dëshiron ta ndryshoj shpejtësinë e ekzekutimit, ai mund të shtyp tastin “1” në tastierë për shpejtësi të madhe, “2” për shpejtësi mesatare dhe “3” për shpejtësi të vogël. Për ta ndërprerë përkohësisht ekzekutimin, përdorim tastin “p” dhe për ta rinisur ekzekutimin, përdorim tastin “r”. Duke klikuar butonin Back përdoruesi kthehet në faqen kryesore për të zgjedhur algoritmin të cilin dëshiron ta vizualizoj.

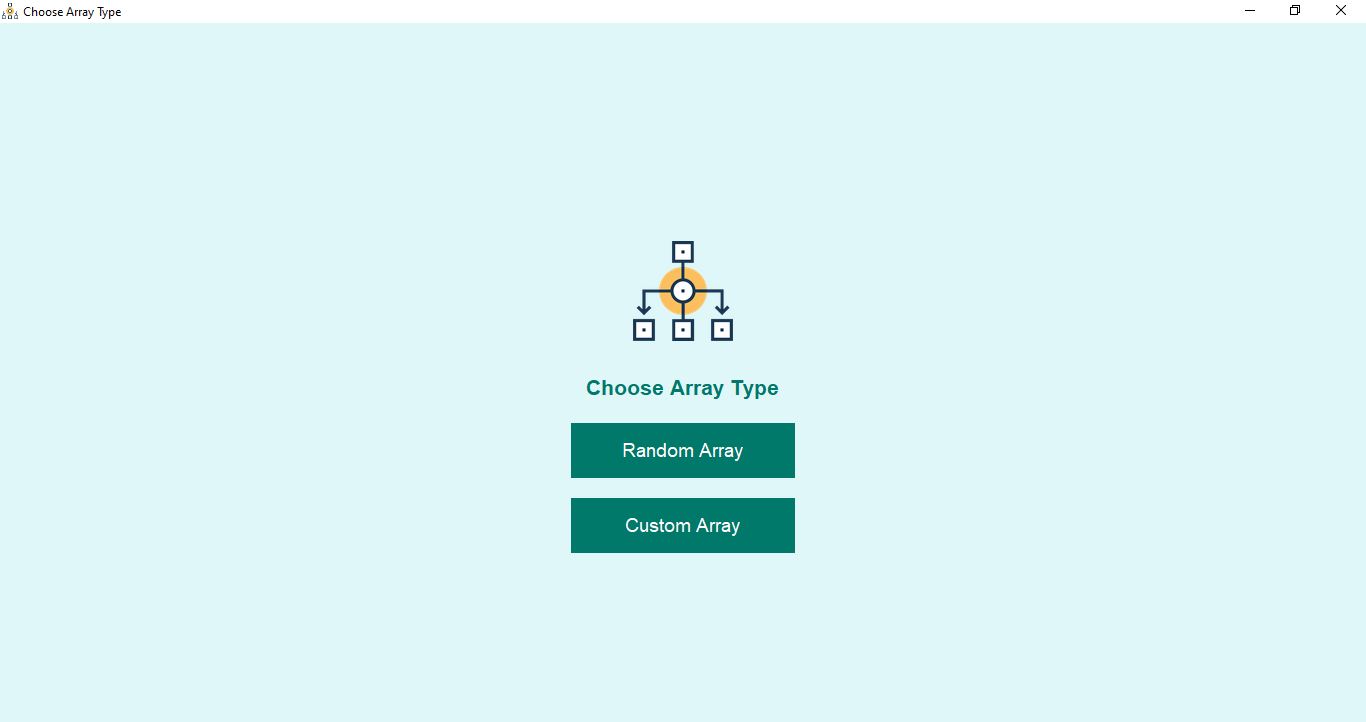


Figura 7 - Faqja për zgjedhjen e tipit të vargut

Pasi përdoruesi kthehet në faqen kryesore dhe zgjedh një algoritëm ai duhet të zgjedh llojin e vargut.

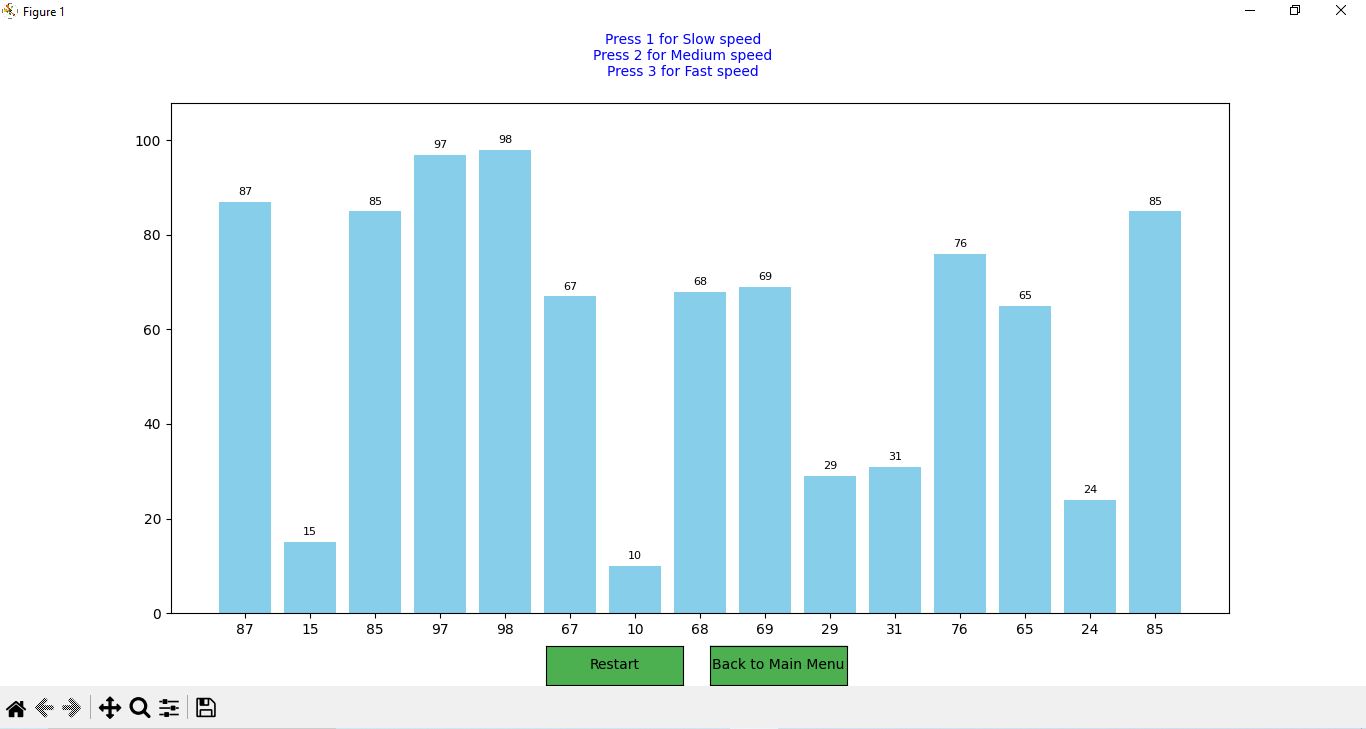


Figura 8 - Pamja iniciale kur zgjedhim një algoritëm dhe vargun e rastësishëm

Kjo figurë tregon rastin kur zgjedhet vargu i rastësishëm, në ekran do të shfaqet një varg me gjatësi dhe elemente të rastësishme. Butoni Restart përdoret nëse përdoruesi dëshiron ta filloj vizualizimin përsëri nga e para kurse Back To Main Menu për t'u kthyer tek fajqa kryesore.



Figura 9 - Zgjedhja e gjatësisë së vargut

Kur përdoruesi zgjedh që të krijoj një varg manualisht, në ekran do të shfaqet dritarja për zgjedhjen e gjatësisë së vargut.

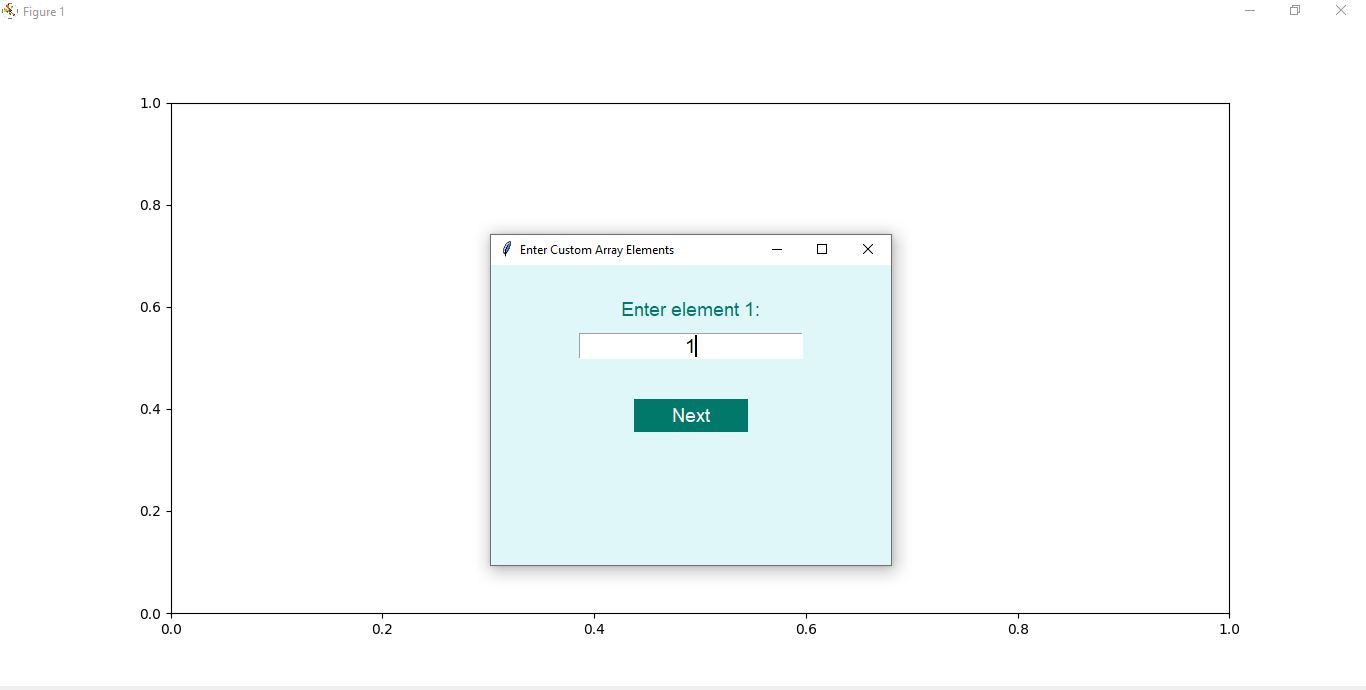


Figura 10 - Zgjedhja e elementeve

Më pas zgjedhen elementet një nga një, në këtë rast katër elemente.

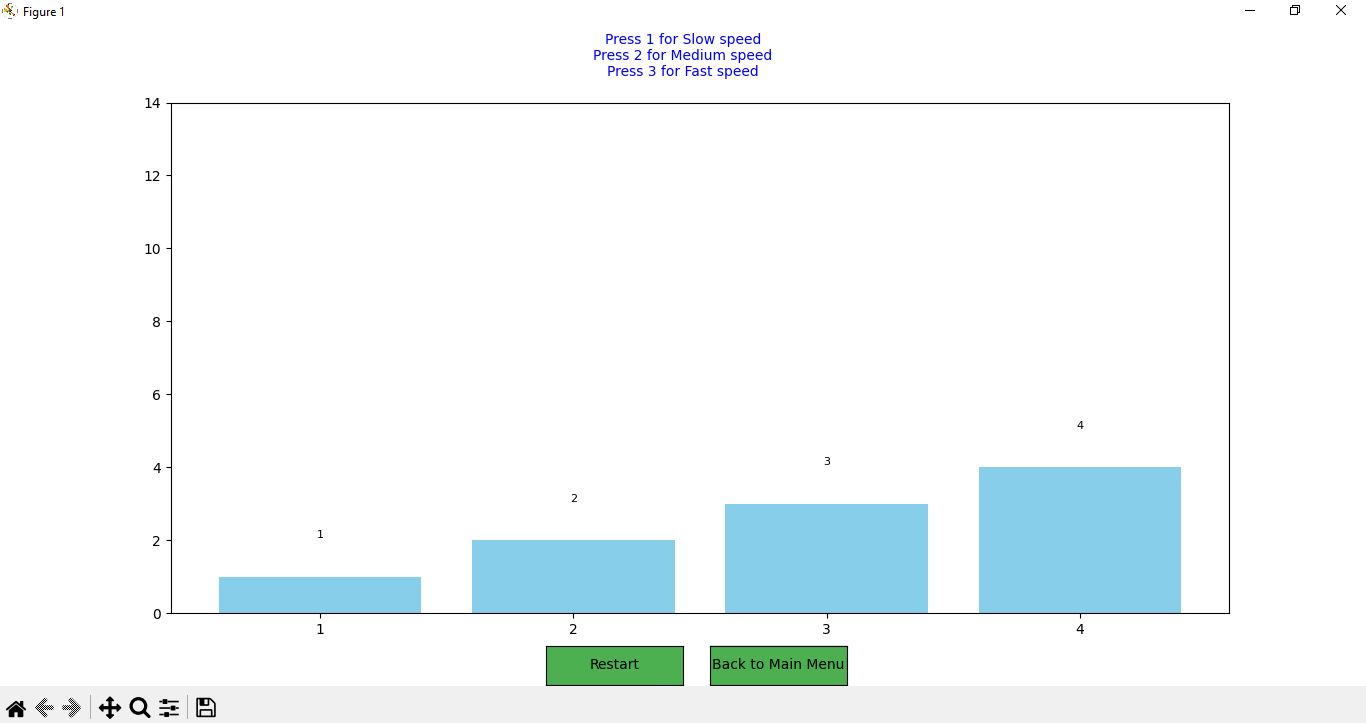


Figura 11 - Vargu manual

Pasi të zgjedhen elementet e vargut, në ekran do të shfaqet kjo dritare me vargun e elementeve të cilat përdoruesi dëshiron t’i vizualizoj.

## Merge Sort

Kur përdoruesi dëshiron të bëj vizualizimin e algoritmit Merge sort, atëherë në faqen kryesore duhet të klikohet butoni Merge Sort dhe të zgjedhet vargu i cili do të vizualizohet. Në këtë shembull do të marret vargu (12,32,23,45,23,51,63).

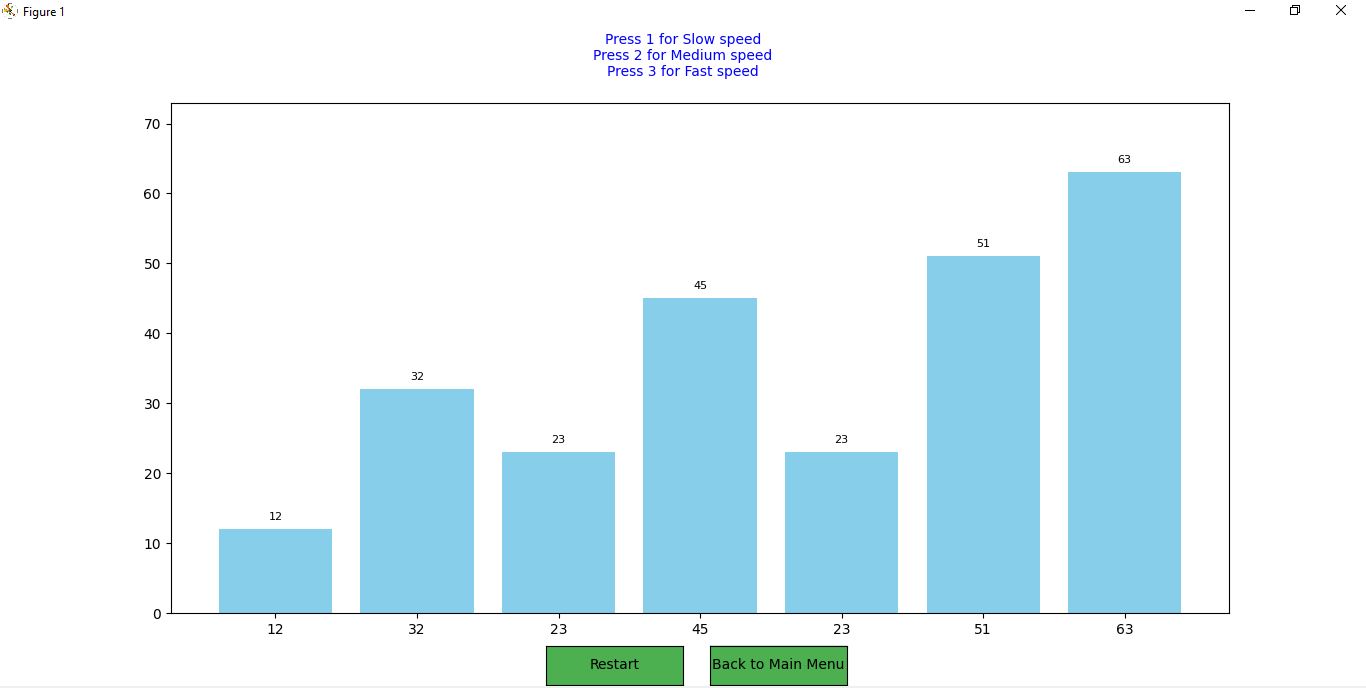


Figura 12 – Pamja e vargut Merge Sort

Pasi të filloj vizualizimi i vargut duke zgjedhur njërin nga opsionet e shpejtësisë, në ekran do të shihen shtyllat të cilat krahasohen në mes vete me ngjyrë të verdhë.

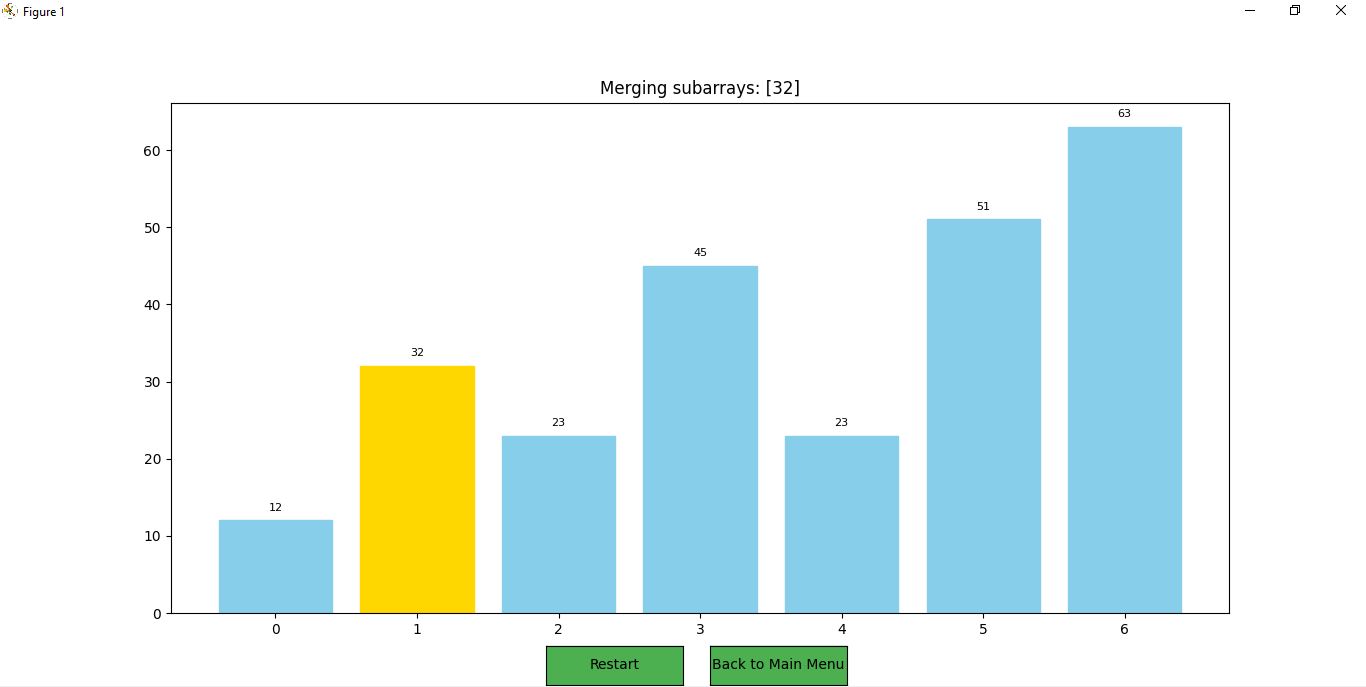
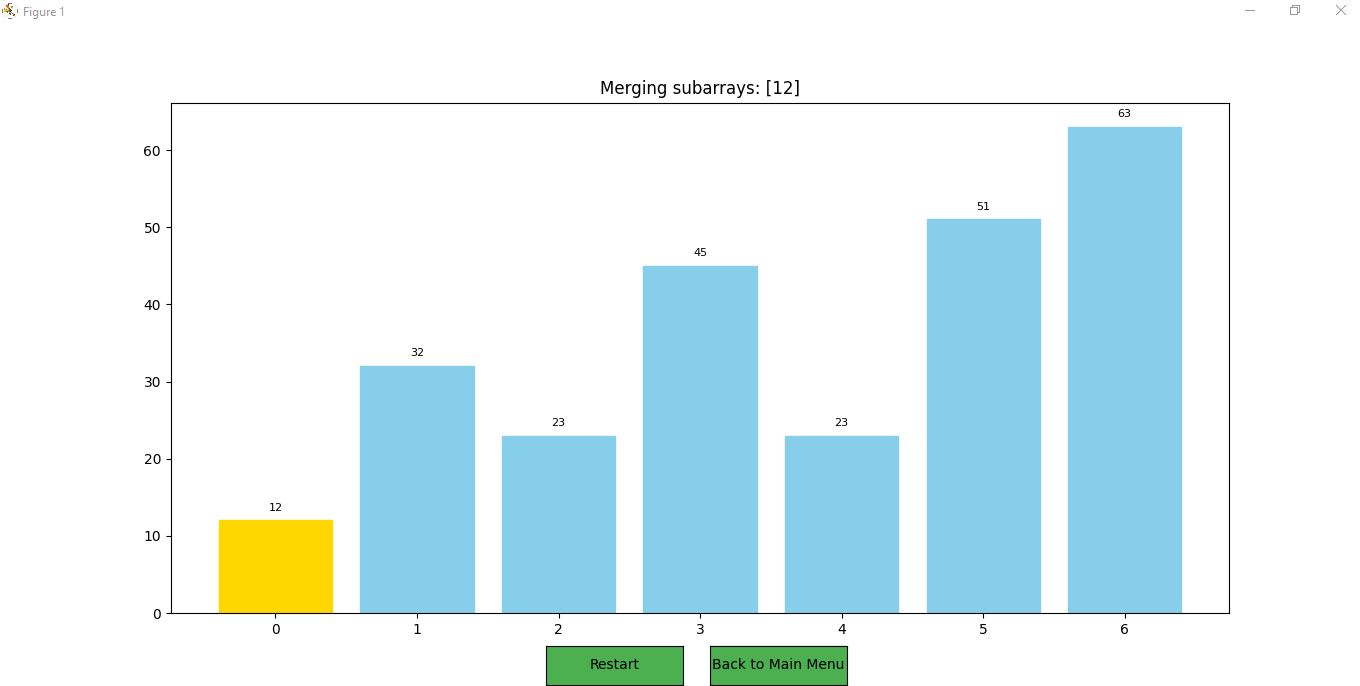


Figura 13 - Krahasimi në mes elementeve

Pas krahasimit nëse ka nevojë për ndërrim vendesh, kryhet ai veprim dhe elementet bashkohen në një nënvarg me ngjyrë të gjelbërt.

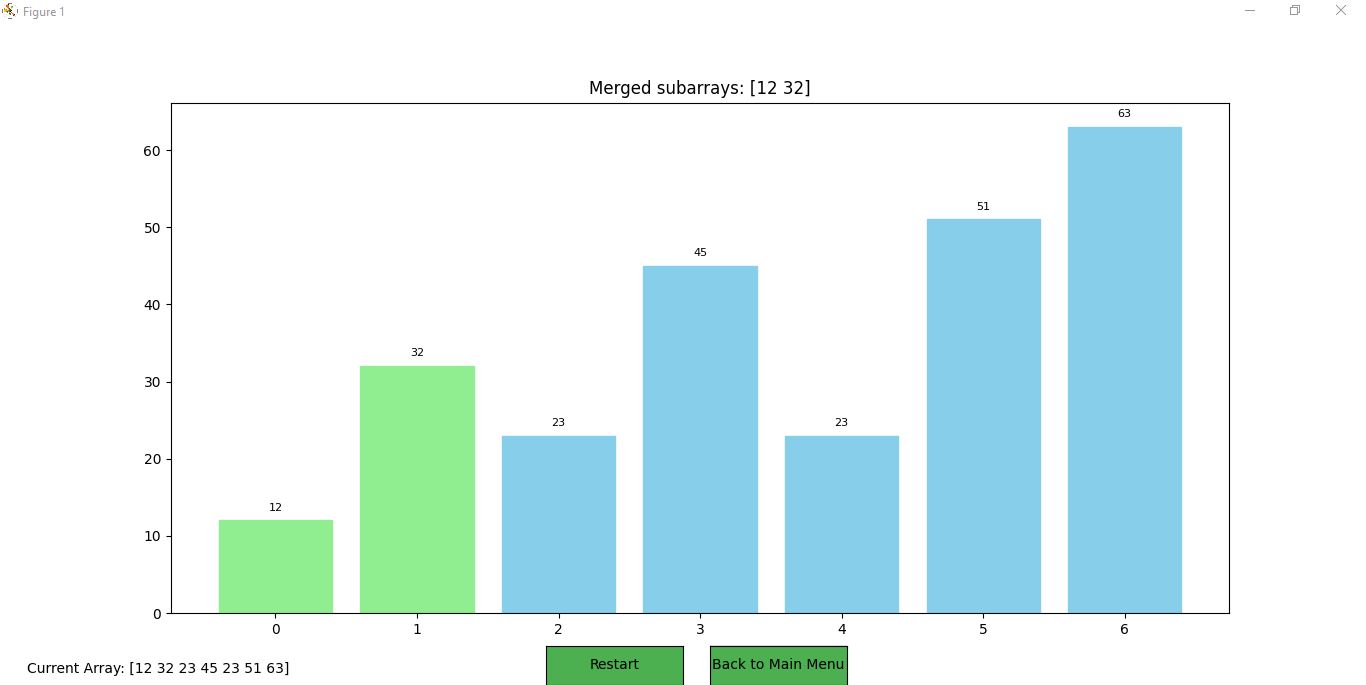


Figura 14 – Nën vargu i parë

Vizualizimi vazhdon duke krahasuar elementet 23 dhe 45 dhe duke i bashkuar si një nënvarg.

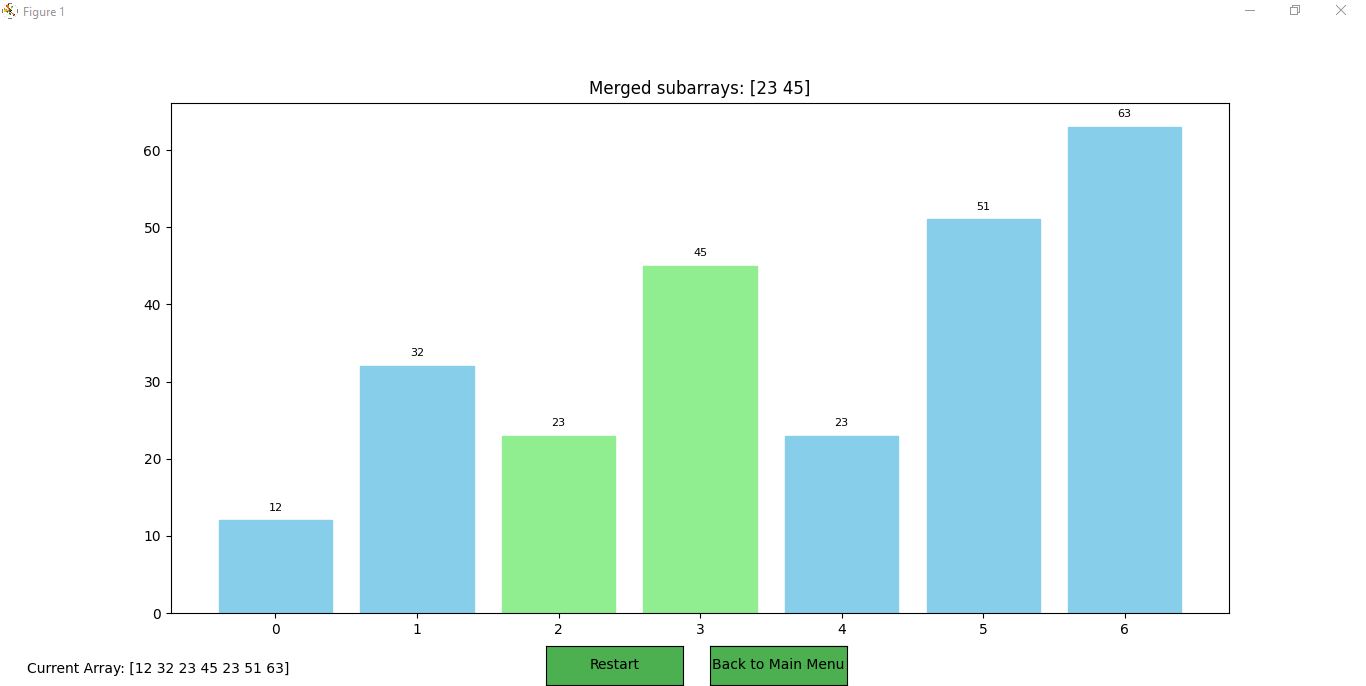


Figura 15 - Nën vargu i dytë

Tani krahasohen dy nëngrupet e krijuara dhe renditen elementet në një grup të vetëm.

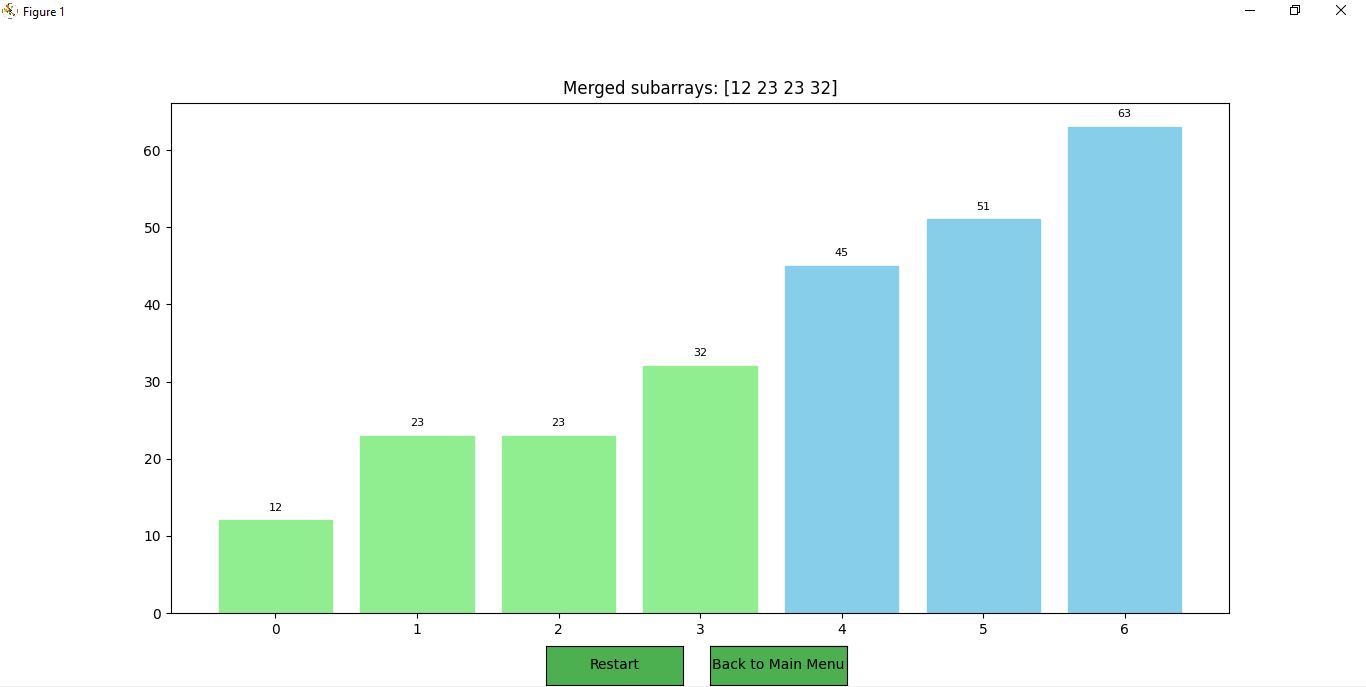


Figura 16 - Bashkimi i dy nën vargjeve

Të njëjtat veprime kryhen fillimisht me elementet 23 dhe 51.

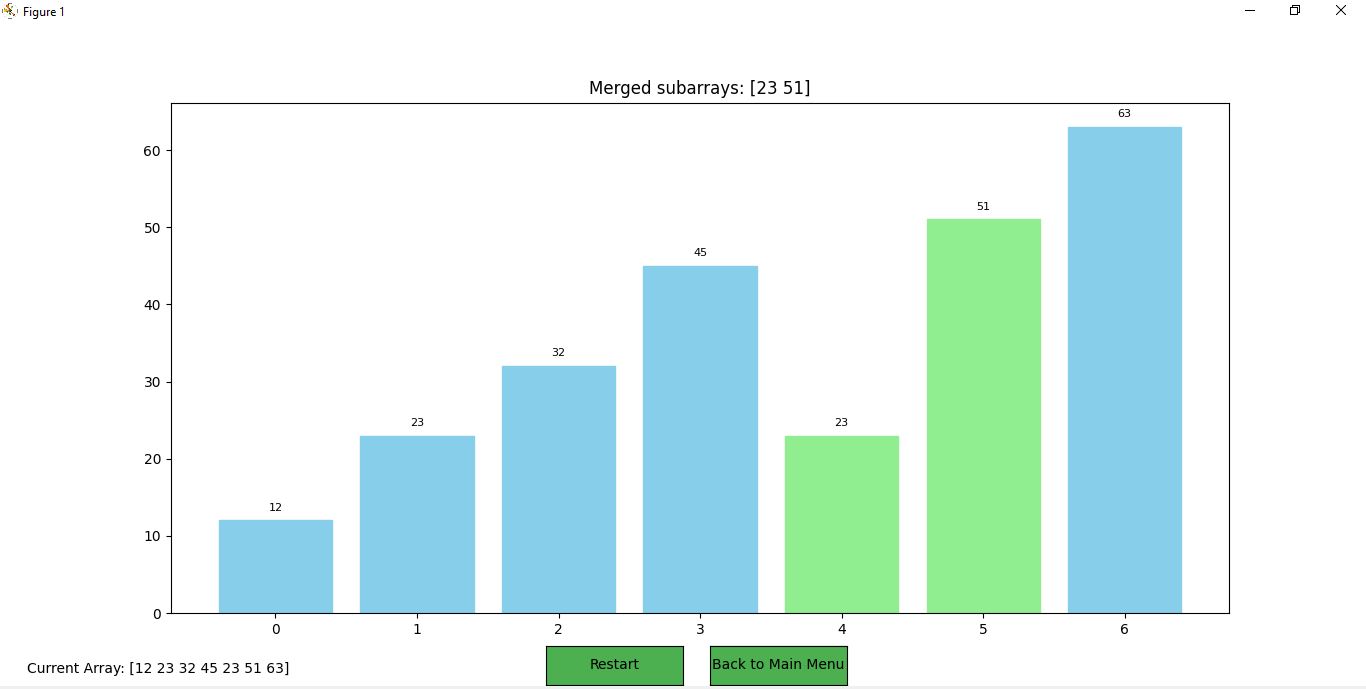


Figura 17 - Nën vagu i tretë

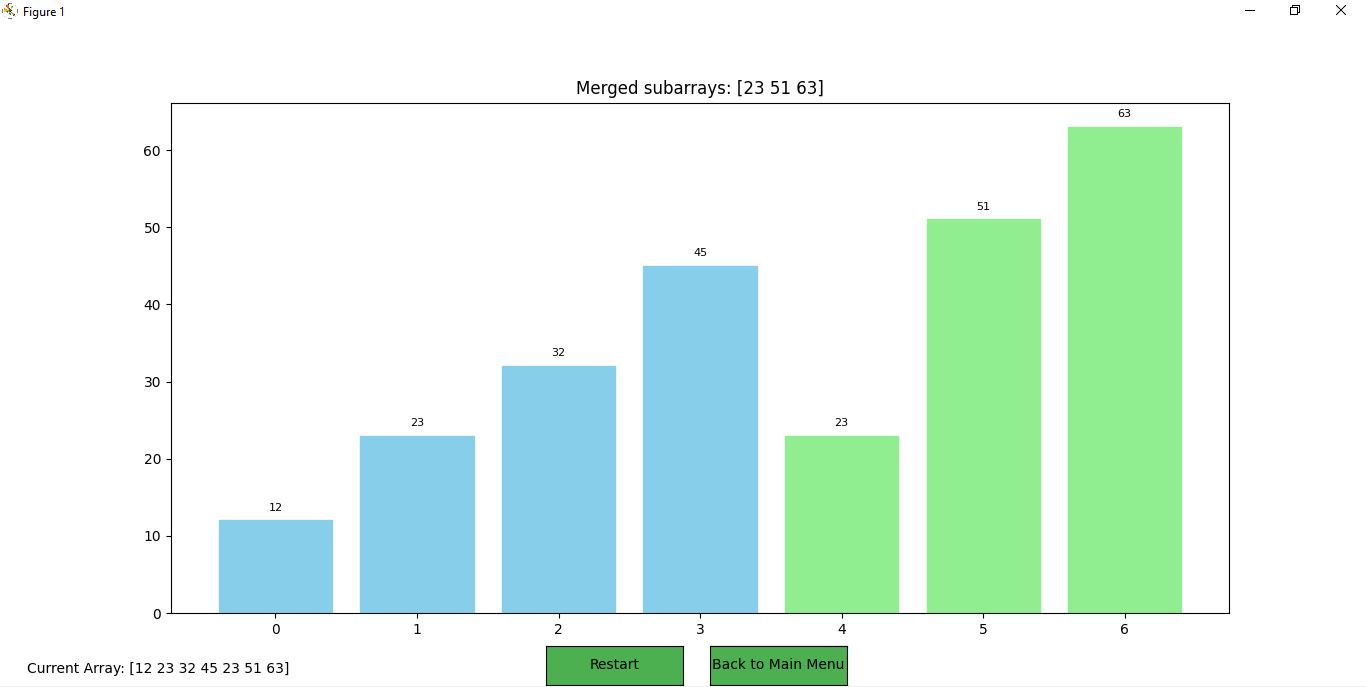


Figura 18 - Bashkimi i nën vagut të tretë dhe të katërt

Dhe më pas nënvargun (23,51) dhe elementin 63.

Si përfundim, dy nën grupe të mëdha (12,23,32,45) dhe (23,51,63) do të bashkohen në një varg të renditur.

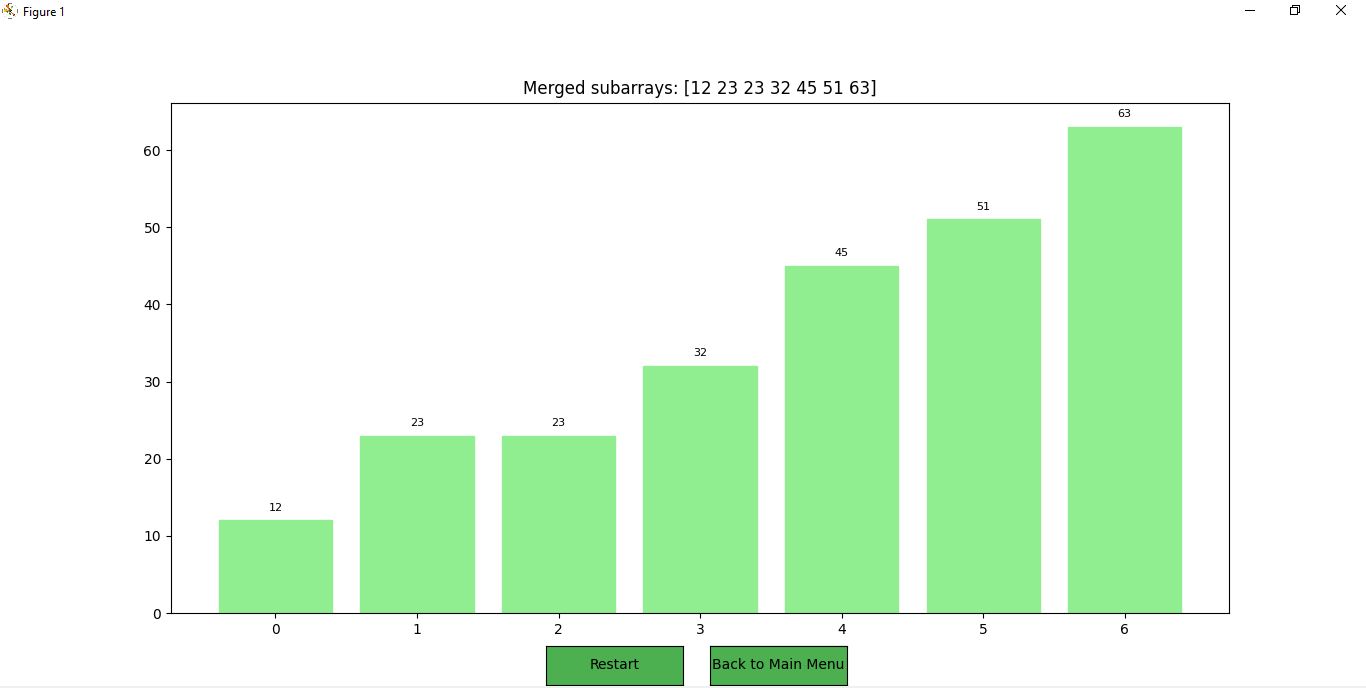


Figura 19 - Sortimi final

Gjatë këtij vizualizimi mund të shihen edhe se cilat elemente janë duke u krahasuar me njëra tjetrën me anë të tekstit sipër grafikut dhe përfundimisht do të shihen nënvargjet e bashkuara në një varg (12,23,23,32,45,51,63).

## Counting Sort

Vargu i cili do të marret si shembull për ta ilustruar counting sort është (5,12,8,3,9,15,7). Fillimisht caktohet range(gamën) të vargut ku range = 15 -3 +1, range = 13.

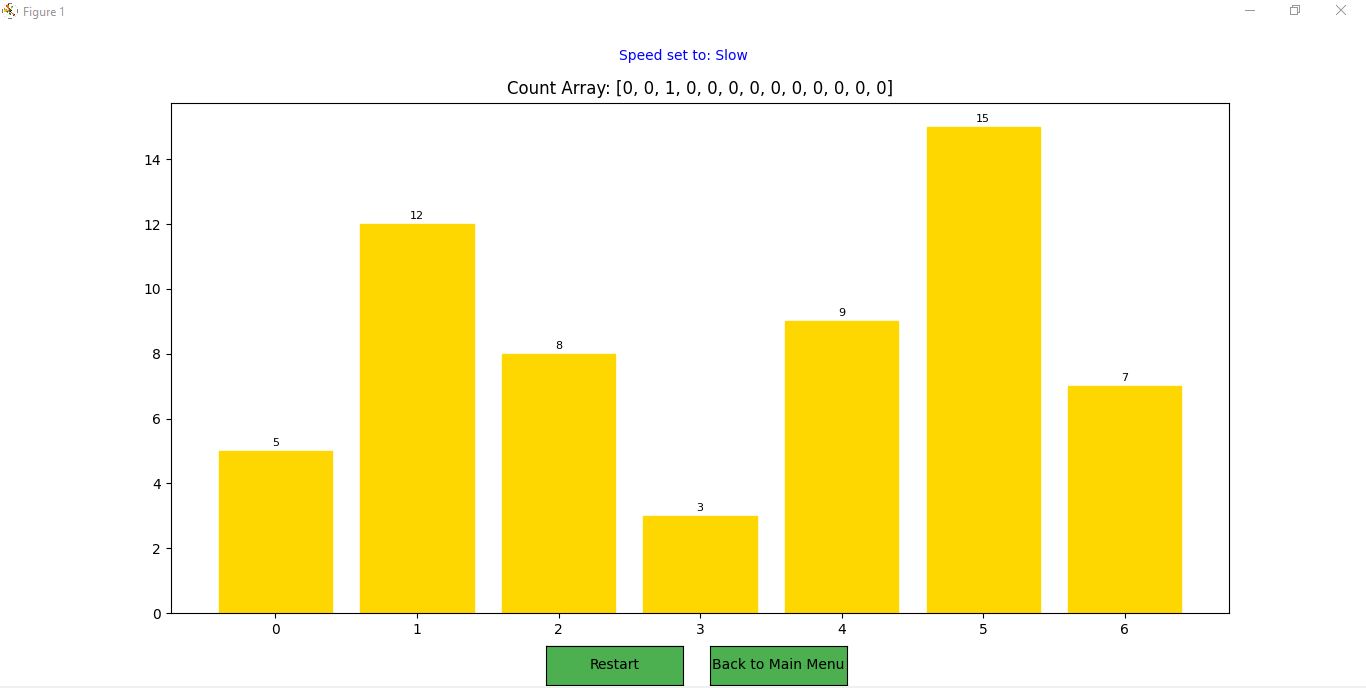


Figura 20 - Pamja fillestare e vargut count

Në ekran do të shfaqet vargu count(numërim) i cili i ka 13 elemente dhe fillon të llogaris count array ku për secilin element llogaritet count(vlera e elementit – elementi më i vogël) +=1. Në figurë shihet count(5-3 =2) +=1 dhe prandaj indeksi 2 është 1.

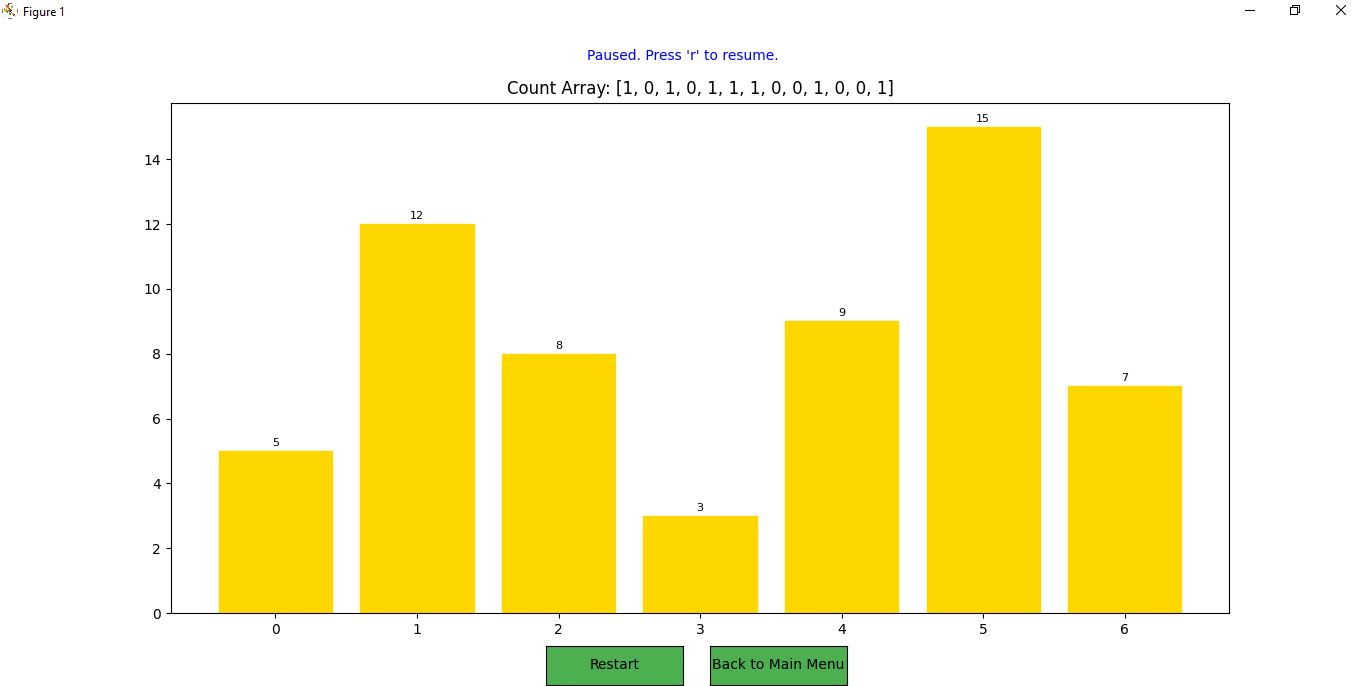


Figura 21 - Pamja finale e vargut count

Aplikacioni e llogarit vargun “count” hap pas hapi për qartësim ndaj përdoruesit.

Kur të përfundoj llogaritja e vargut count ai do të ketë vlerën (1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1)

Hapi i radhës është llogaritja e vargut komulativ, me ç’rast përdoruesi edhe njoftohet në ekran. Mënyra se si funksionon comulative array është kjo cumulative(0) = count(0) = 1, cumulative(1) = cumulative(0) + count(1) = 1 + 0 = 1 ku edhe është ndërrimi i parë i tij prej vargut count.

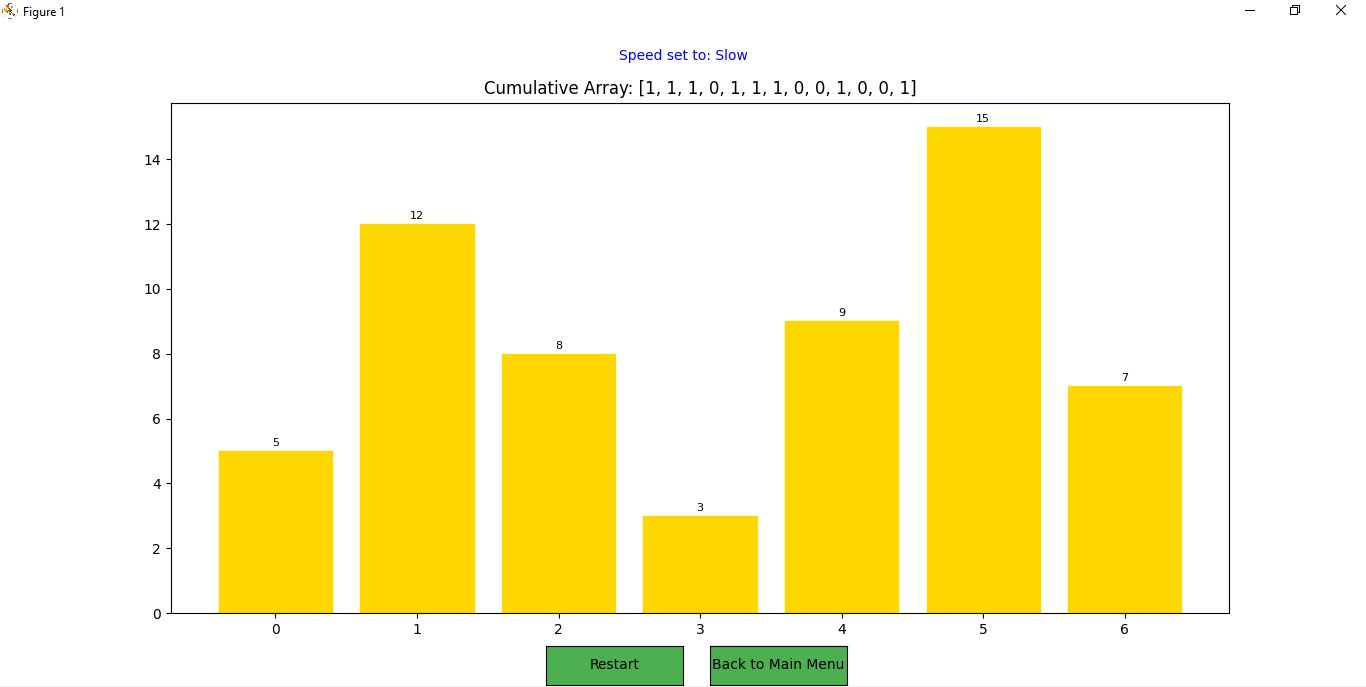


Figura 22 - Pamja fillestare e vargut cumulative

Më pas kemi cumulative (2) = cumulative(1) + count(2) = 1 + 1 = 2, cumulative(3) = cumulative(2) + count(3) = 2 + 0 = 2 dhe llogaritjen e vazhdojnë deri sa vargu plotësohet cumulative = (1, 1, 2, 2, 3, 4, 5, 5, 5, 6, 6, 6, 7).

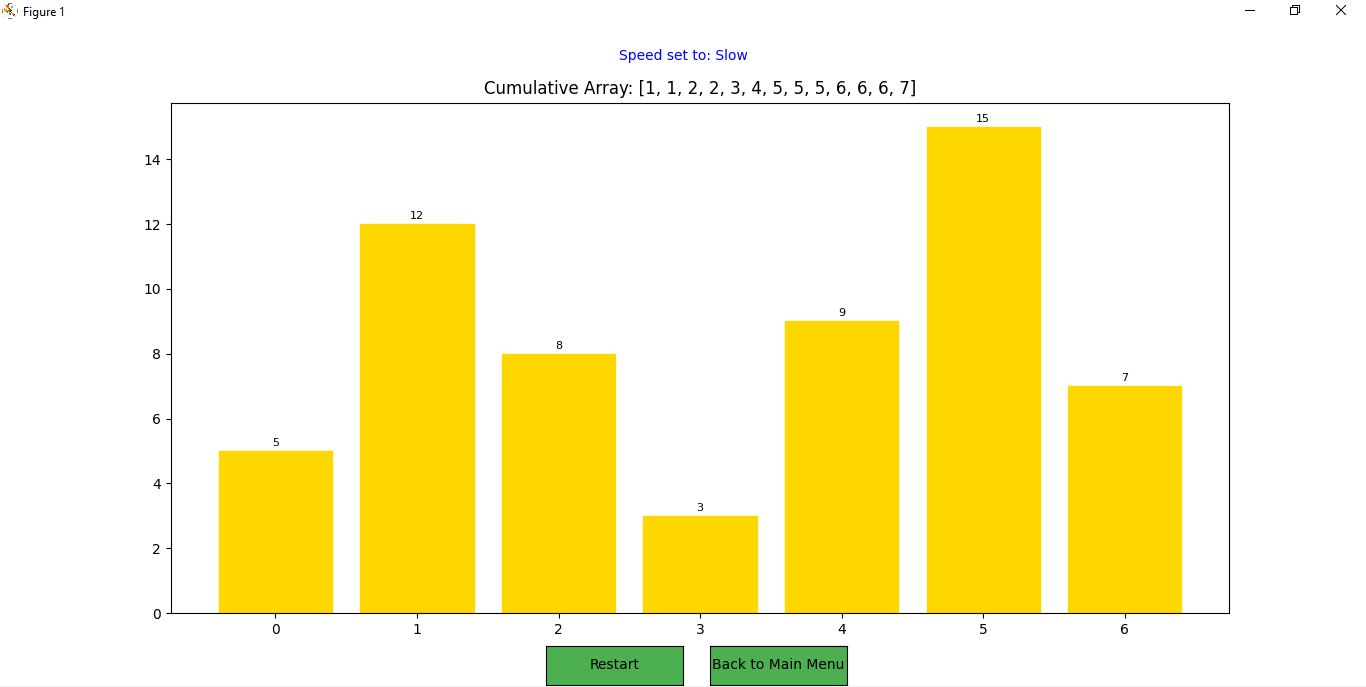


Figura 23 - Pamja finale e vargut cumulative

Tani, duke përdorur vargun cumulative fillohet ndërrimi i vendeve të elementeve. Duke filluar nga elementi i fundit i vargut fillestar (5,12,8,3,9,15,7) merret 7 dhe zbritet me elementin më të vogël 3, pastaj merret comulative(4) = 3 që do të thotë se 7 duhet të vendoset tek indeksi 2 i vargut të sortuar, më pas vazhdohet me 15 e kështu me radhë.

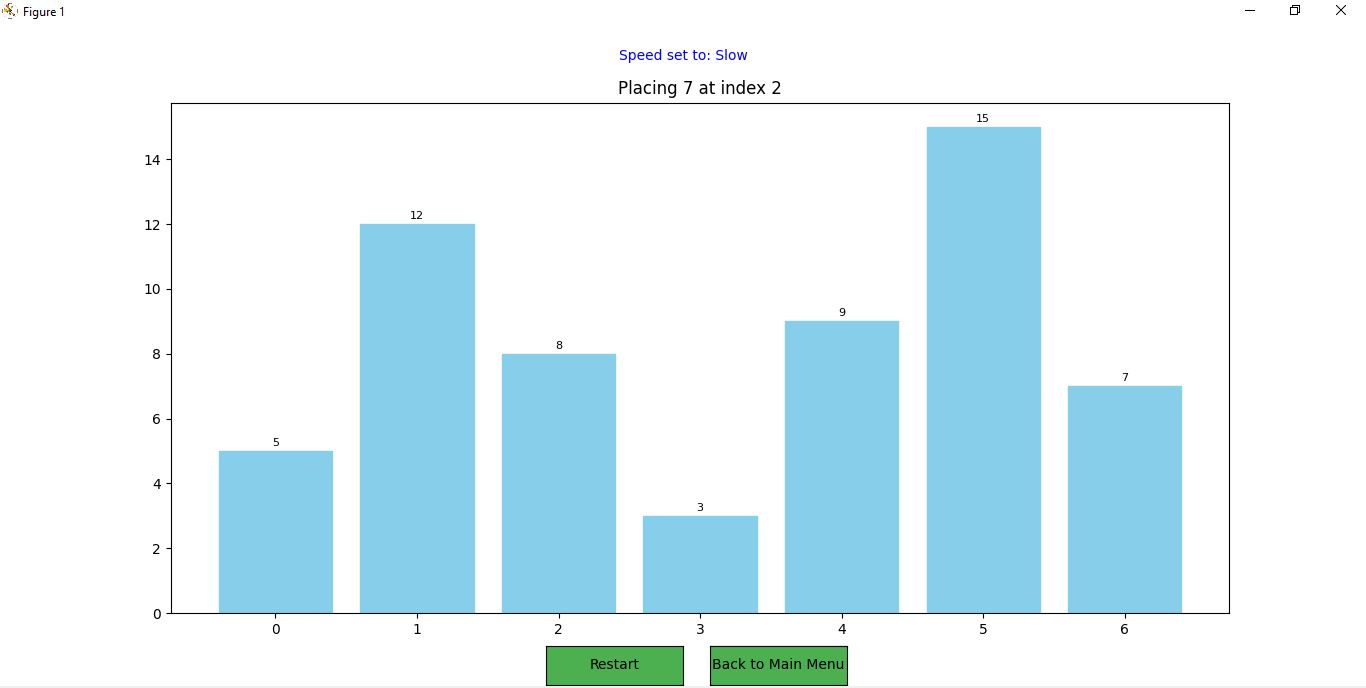


Figura 24 - Shfaqja e indeksit të elementeve në ekran

Përdoruesi lajmërohet për këto ndërrime në ekran paraprakisht dhe më pas fillon ndërrimi i vendeve të elementeve.

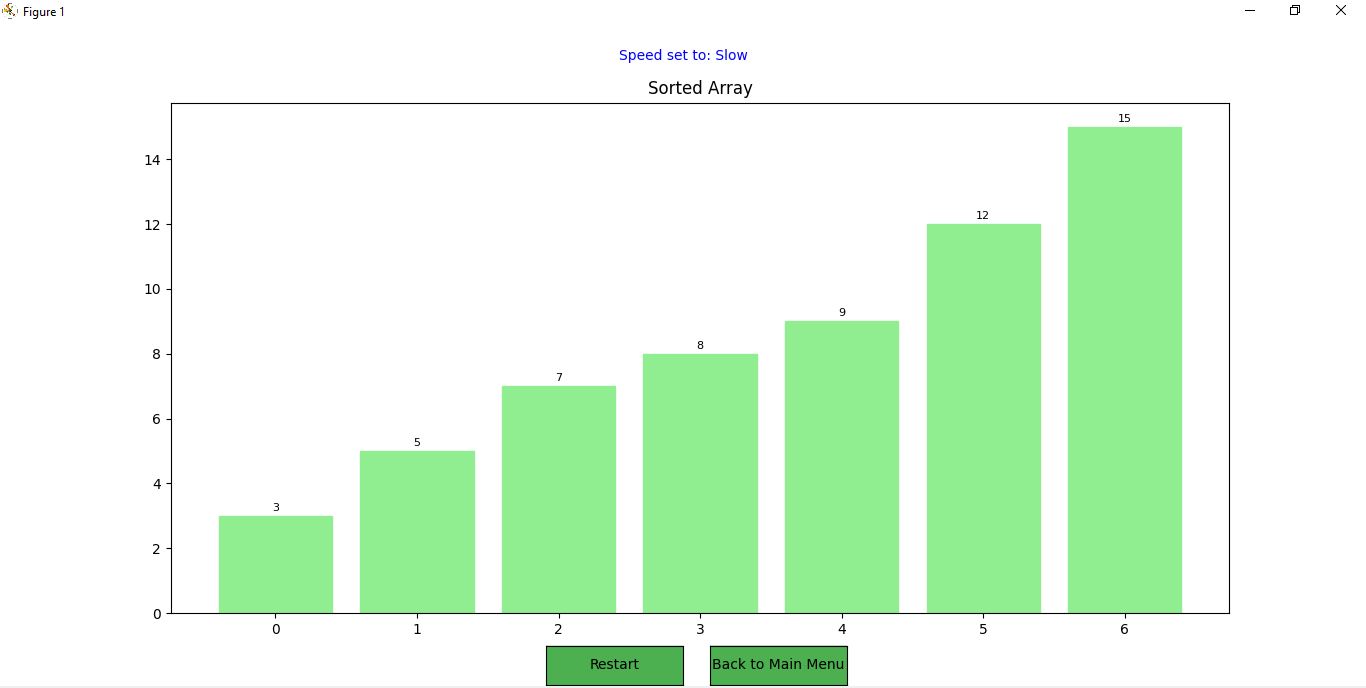


Figura 25 - Vargu i sortuar

## Bucket Sort

Tek algoritmi bucket sort aplikacioni i ka 10 kova të cilat mbushen me elementet të cilat i ka vargu, kova 1 i merr elementet prej 0-9, kova 2 elementet 10-19, kova 3 elementet 20-29... kova 10 elementet 9-99 prandaj ekziston një limit i madhësisë së elementeve të cilat përdoruesi mund t’i përdorë.

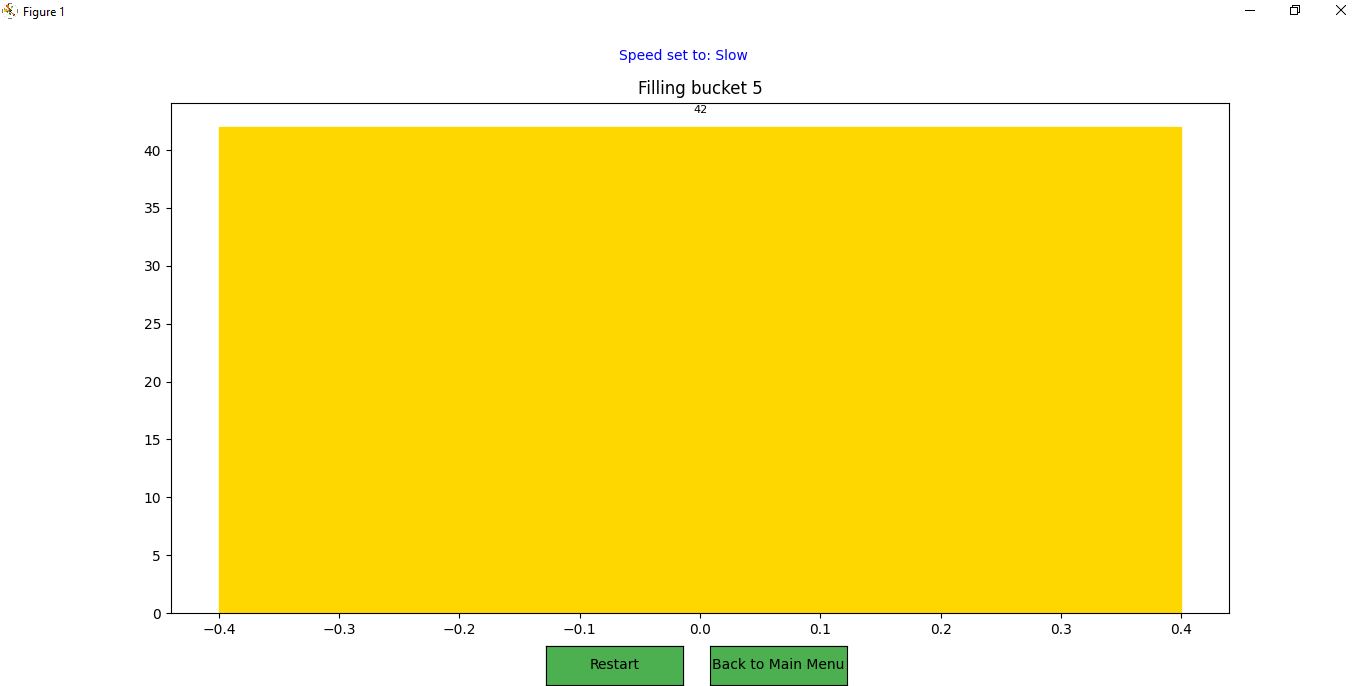


Figura 26 - Elementi i parë shtohet në kovën përkatëse

Kur fillon ekzekutimi i aplikacionit hapi i parë është mbushja e kovave duke filluar nga elementi i parë i vargut, siç shihet elementi 42 i takon kovës 5.

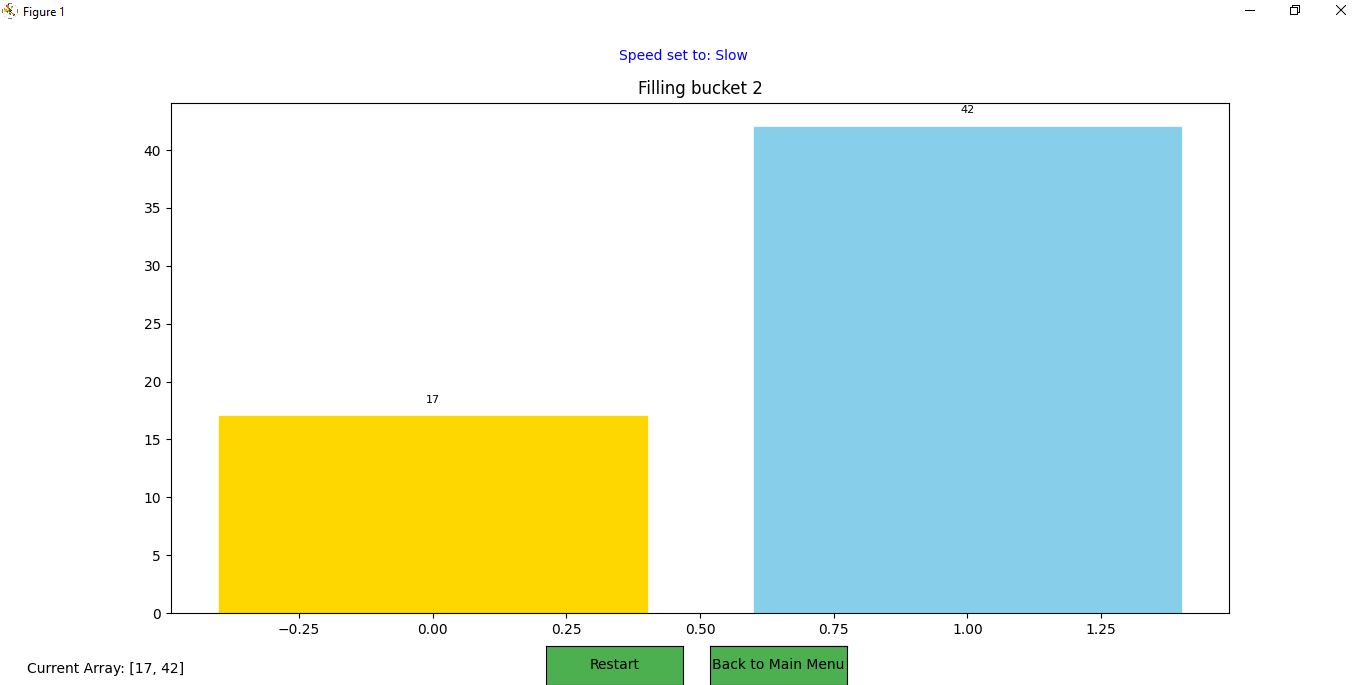


Figura 27 - Elementi i dytë shtohet në kovën përkatëse

Më pas elementi 17 shtohet në kovën 2 e kështu me radhë.

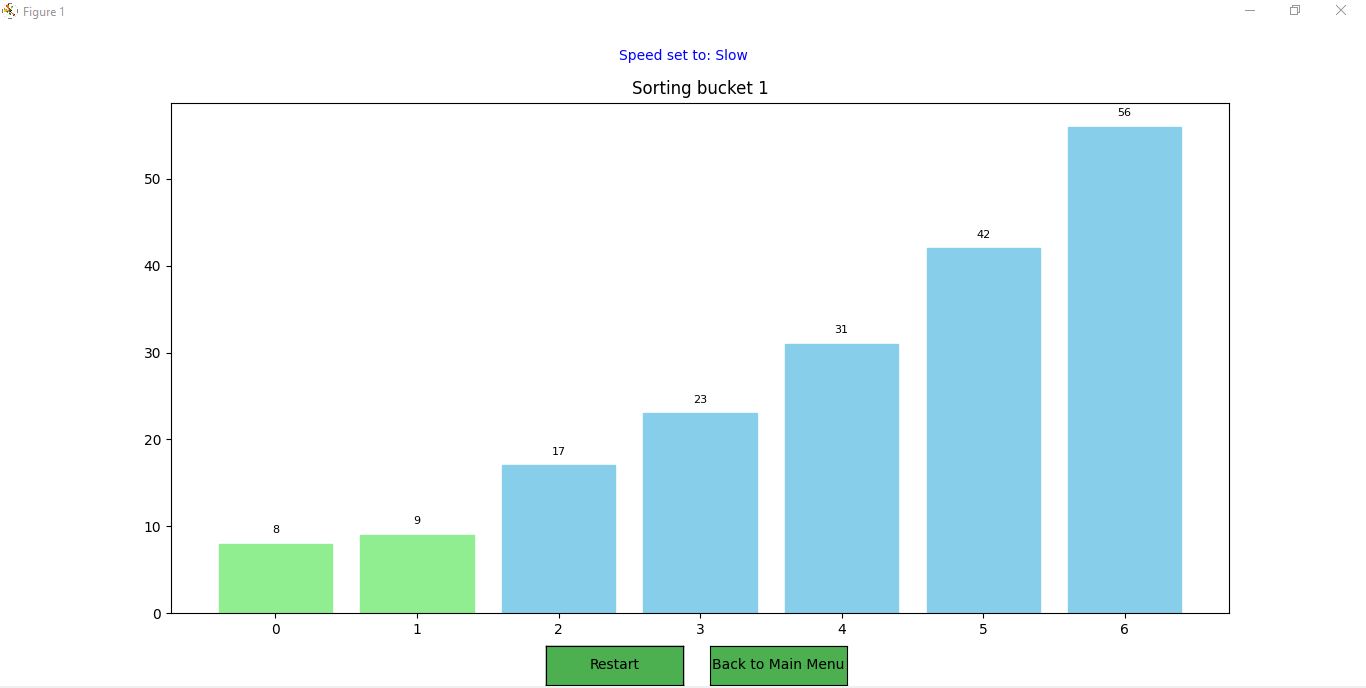


Figura 28 - Sortimi i elementeve brenda kovave

Pas shtimit të elementeve nëpër kova fillon sortimi i kovave individualisht ku bëhet ndërrimi i vendeve të elementeve sipas madhësisë. Ky sortim vazhdon deri tek kova numër 10 edhe nëse nuk ka elemente në të.

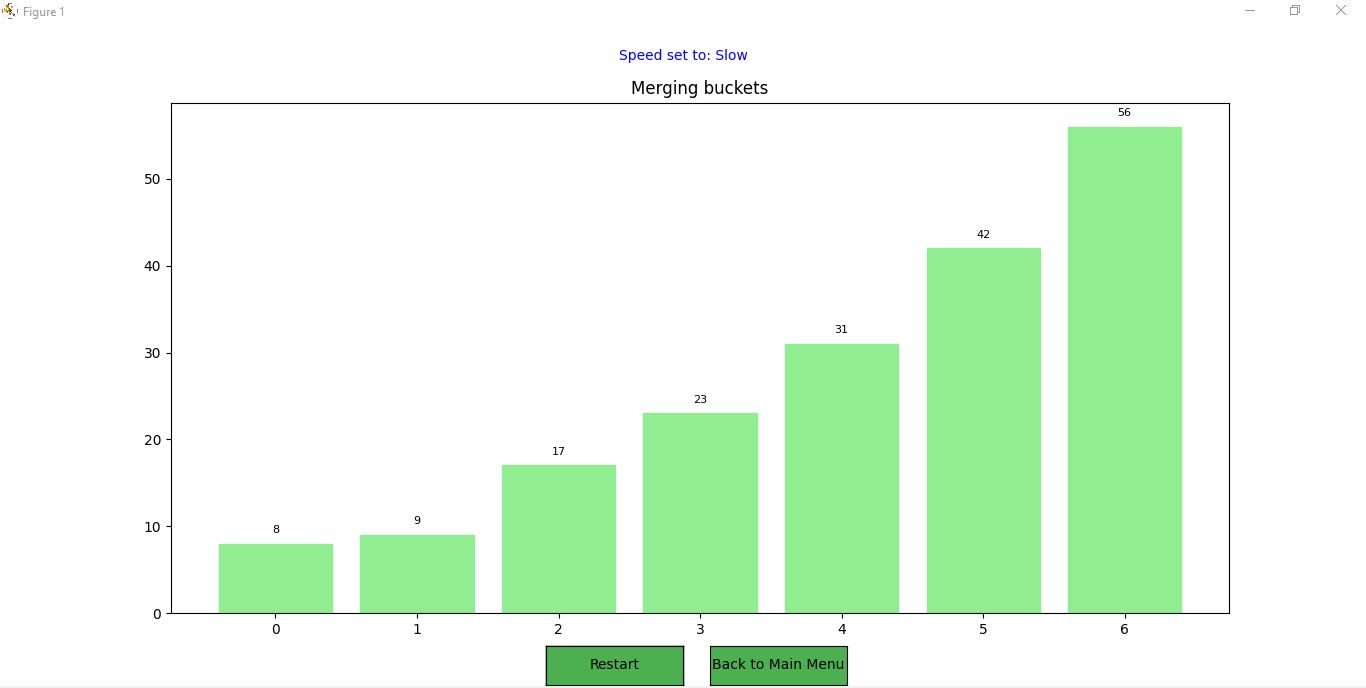


Figura 29 - Bashkimi i kovave pas sotimit

Pas sortimit nëpër kova ato bashkohen duke krijuar vargun e sortuar.

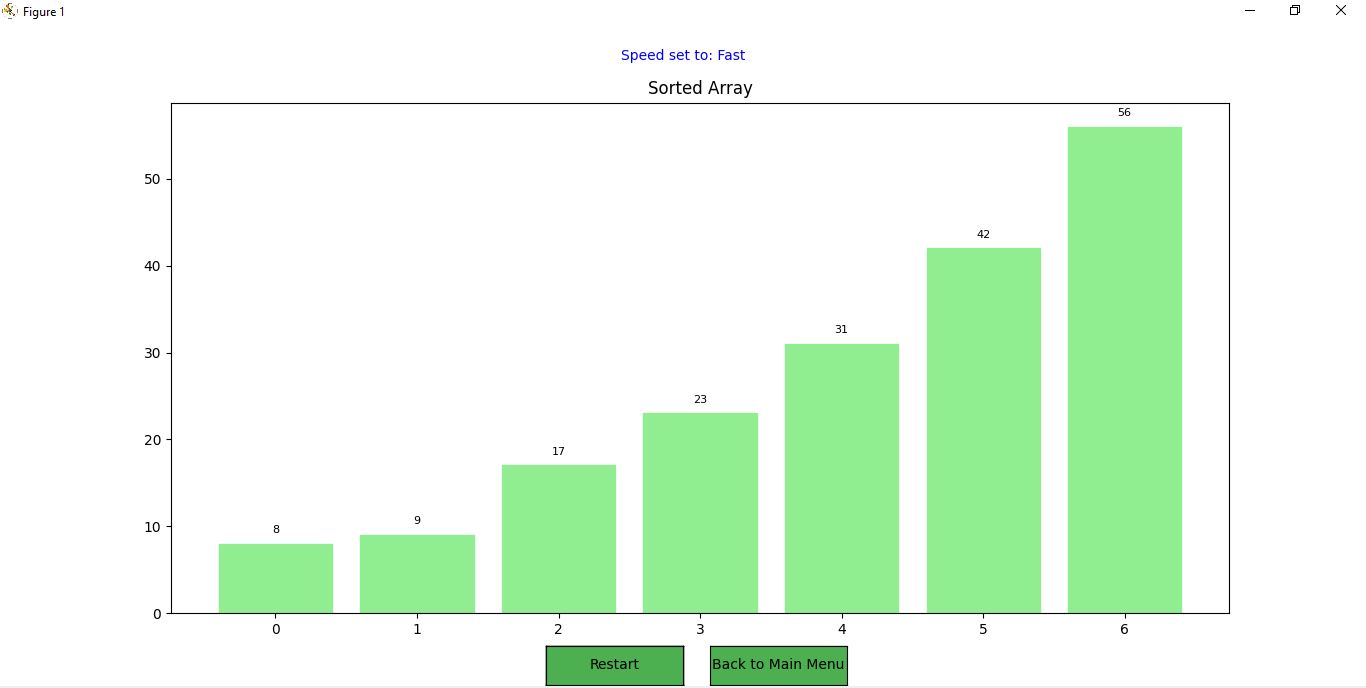


Figura 30 - Vargu i sortuar

Pas bashkimit të kovave në ekran merret konfirmim “sorted array”.

# Përfundimet

Në këtë punim të diplomës është bërë një studim i tre algoritmeve të sortimit Merge Sort, Counting Sort dhe Bucket Sort, krahasimi i performancës së tyre si dhe krijimi i një aplikacioni për vizualizimin e tyre duke përdorur gjuhën programuese Python. Aplikacioni është mjaft i dobishëm për studentët e interesuar të mësojnë se si punojnë këto algoritme gjatë sortimit të vargjeve sidomos për arsyen se gjatë vizualizimit ka edhe shpjegime se çfarë hapash po ekzekutohen.

Algoritmet e sortimit janë të dobishëm për menaxhimin e të dhënave në sisteme me volum të madh të të dhënave, ata përdoren edhe për përcaktimin e prioritizimit të punëve që duhen kryer.

Algoritmi Merge Sort është një algoritëm sortii i cili veçanti e ka stabilitetin e tij i cili është ndër arsyet kryesore pse përdoret. Ky algoritëm e ndan vargun hap pas hapi deri kur mbesin vetëm nga një element dhe më pas fillon duke i bashkuar nënvargjet dhe sortuar ato.

Algoritmi Counting Sort i ka disa hapa për sortimin e elementeve të vargut të cilat përfshijnë: numërimin e frekuencave, akumulimin, renditjen e elementeve dhe kopjimin e elementeve.

Algoritmi Bucket Sort i ndan elementet në kova dhe fillimisht i rendit ato elemente brenda kovave dhe në fund i bashkon kovat duke krijuar një varg të vetëm.

Për të pasqyruar dallimet mes performancave të këtyre algoritmeve shihet tabela e mëposhtme: (1)

|  |  |  |  |
| --- | --- | --- | --- |
| Algoritmi | Rasti më i mirë | Rasti mesatar | Rasti më i keq |
| Merge Sort | O(n log n) | O(n log n) | O(n log n) |
| Counting Sort | O(n + k) | O(n + k) | O(n + k) |
| Bucket Sort | O(n + k) | O(n + k) | O() |

Si përfundim arrihet të konstatohet se secili algoritëm i ka karakteristikat e veta, avantazhet dhe disavantazhet dhe nëse dëshirojmë të kemi përgjigjjen e pyetjes se cili prej këtyre algoritmeve është i duhuri nuk do kemi përgjigjje të saktë, kjo tërësisht varet nga teksti në të cilin po zhvillohet kërkimi dhe nga mostra e cila po kërkohet.

# Summary

In this thesis, a study has been conducted on three sorting algorithms: Merge Sort, Counting Sort, and Bucket Sort, comparing their performance and creating an application for their visualization using the Python programming language. The application is quite useful for students interested in learning how these algorithms work during array sorting, especially because, during the visualization, there are also explanations of the steps being executed.

Sorting algorithms are useful for data management in systems with large volumes of data, and they are also used for determining the prioritization of tasks that need to be executed.

The Merge Sort algorithm is a sorting algorithm that is particularly known for its stability, which is one of the main reasons it is used. This algorithm divides the array step by step until only one element remains in each subarray, and then it starts merging and sorting those subarrays.

The Counting Sort algorithm has several steps for sorting the elements of the array, which include counting the frequencies, accumulating, sorting the elements, and copying them.

The Bucket Sort algorithm divides the elements into buckets, initially sorting the elements within the buckets, and finally merging the buckets to create a single array.

To reflect the differences in the performance of these algorithms, we present the following table: (8)

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Best Case | Average Case | Worst Case |
| Merge Sort | O(n log n) | O(n log n) | O(n log n) |
| Counting Sort | O(n + k) | O(n + k) | O(n + k) |
| Bucket Sort | O(n + k) | O(n + k) | O() |

In conclusion each algorithm has its own characteristics, advantages and disadvantages and if we want to know which of these algorithms is the right one to choose we will not have the correct answer, it depends entirely from the text and from the sample which we are looking for.

# Shtesat

**Lista e Figurave**

[Figura 1 - Algoritmet krahasuese dhe algoritmet jo krahasuese 10](#_Toc175912055)

[Figura 2 - Ilustrim i Merge Sort 12](#_Toc175912056)

[Figura 3 - Ilustrim i Couting Sort 15](#_Toc175912057)

[Figura 4 - Ilustrim i Bucket Sort 17](#_Toc175912058)

[Figura 5 - Faqja kryesore e aplikacionit 22](#_Toc175912059)

[Figura 6 - Faqja e ndihmës 23](#_Toc175912060)

[Figura 7 - Faqja për zgjedhjen e tipit të vargut 24](#_Toc175912061)

[Figura 8 - Pamja iniciale kur zgjedhim një algoritëm dhe vargun e rastësishëm 25](#_Toc175912062)

[Figura 9 - Zgjedhja e gjatësisë së vargut 25](#_Toc175912063)

[Figura 10 - Zgjedhja e elementeve 26](#_Toc175912064)

[Figura 11 - Vargu manual 26](#_Toc175912065)

[Figura 12 – Pamja e vargut Merge Sort 27](#_Toc175912066)

[Figura 13 - Krahasimi në mes elementeve 27](#_Toc175912067)

[Figura 14 – Nën vargu i parë 28](#_Toc175912068)

[Figura 15 - Nën vargu i dytë 28](#_Toc175912069)

[Figura 16 - Bashkimi i dy nën vargjeve 29](#_Toc175912070)

[Figura 17 - Nën vagu i tretë 29](#_Toc175912071)

[Figura 18 - Bashkimi i nën vagut të tretë dhe të katërt 30](#_Toc175912072)

[Figura 19 - Sortimi final 30](#_Toc175912073)

[Figura 20 - Pamja fillestare e vargut count 31](#_Toc175912074)

[Figura 21 - Pamja finale e vargut count 31](#_Toc175912075)

[Figura 22 - Pamja fillestare e vargut cumulative 32](#_Toc175912076)

[Figura 23 - Pamja finale e vargut cumulative 32](#_Toc175912077)

[Figura 24 - Shfaqja e indeksit të elementeve në ekran 33](#_Toc175912078)

[Figura 25 - Vargu i sortuar 33](#_Toc175912079)

[Figura 26 - Elementi i parë shtohet në kovën përkatëse 34](#_Toc175912080)

[Figura 27 - Elementi i dytë shtohet në kovën përkatëse 35](#_Toc175912081)

[Figura 28 - Sortimi i elementeve brenda kovave 35](#_Toc175912082)

[Figura 29 - Bashkimi i kovave pas sotimit 36](#_Toc175912083)

[Figura 30 - Vargu i sortuar 36](#_Toc175912084)

# Referencat

1. ***International institue in Geneva.* [Në linjë] [Cituar më: 21 Korrik 2024.] https://www.iig.ch/en-en/blog/computer-science/algorithm-computer-science-definition-and-understanding#:~:text=In%20Computer%20Science%2C%20an%20algorithm,the%20understanding%20of%20available%20alternatives..**

**2. Geneva G. Belford, Allen Tucker. *Britannica.* [Në linjë] [Cituar më: 21 Korrik 2024.] https://www.britannica.com/science/computer-science/Algorithms-and-complexity.**

**3. *DataCamp.* [Në linjë] [Cituar më: 23 Korrik 2024.] https://www.datacamp.com/blog/what-is-an-algorithm.**

**4. What are the best algorithms for specific use cases? [Në linjë] [Cituar më: 6 Gusht 2024.] https://www.linkedin.com/advice/3/what-best-algorithms-specific-use-cases-skills-algorithms-2c4lf.**

**5. McConnell, Jeffrey J. *Analysis of Algorithms: An Active Learning Approach.***