Sortari Structuri de date

Ghiuzan Edward

Grupa 331

Sortarile alese:

- Radix sort
- Merge sort
- Shell sort
- Count sort
- Quick Sort
- Bucket Sort

Radix Sort

Ce este algoritmul de sortare Radix?

Radix Sort este un algoritm de sortare non-comparativ. Funcționează prin gruparea cifrelor individuale ale elementelor de sortat. O tehnică de sortare stabilă este apoi utilizată pentru a organiza elementele pe baza rădăcinii lor. Este un algoritm de sortare liniară.

Pașii Algoritmului Radix Sort:

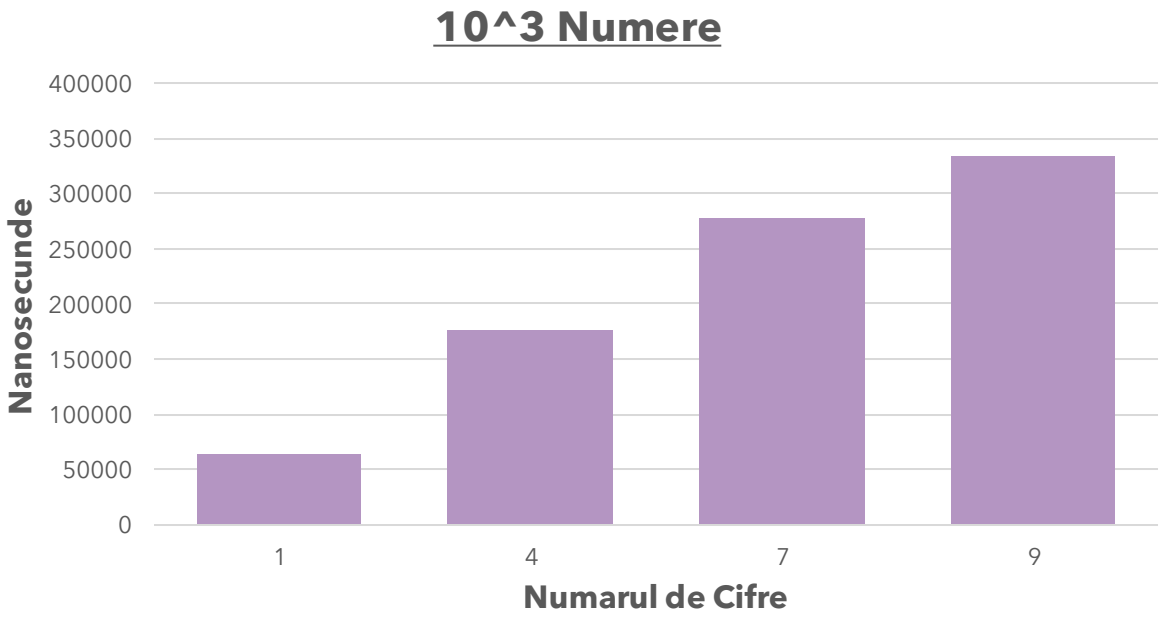
- Găsirea elementului maxim și obținerea numărului de cifre ale acelui element. Ne oferă numărul de iterații pe care le va urma procesul de sortare.
- Grupați cifrele individuale ale elementelor în aceeași poziție semnificativă în fiecare iterație.
- Procesul de grupare va începe de la cifra cea mai puțin semnificativă și se va termina cu cifra cea mai semnificativă.
- Sortarea elementelor pe baza cifrelor din acea poziție semnificativă.
- Menținerea ordinii relative a elementelor care au aceeași valoare a cheii. Această proprietate a sortării radix îl face un sortare stabil.

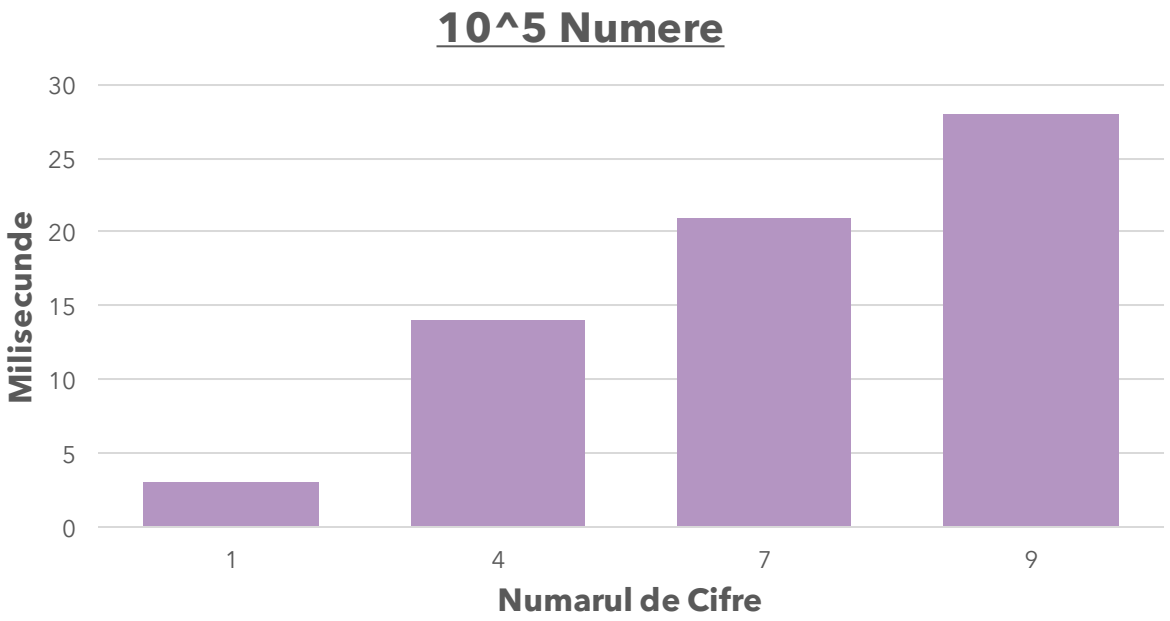
  Iterația finală ne va oferi o listă complet sortată.

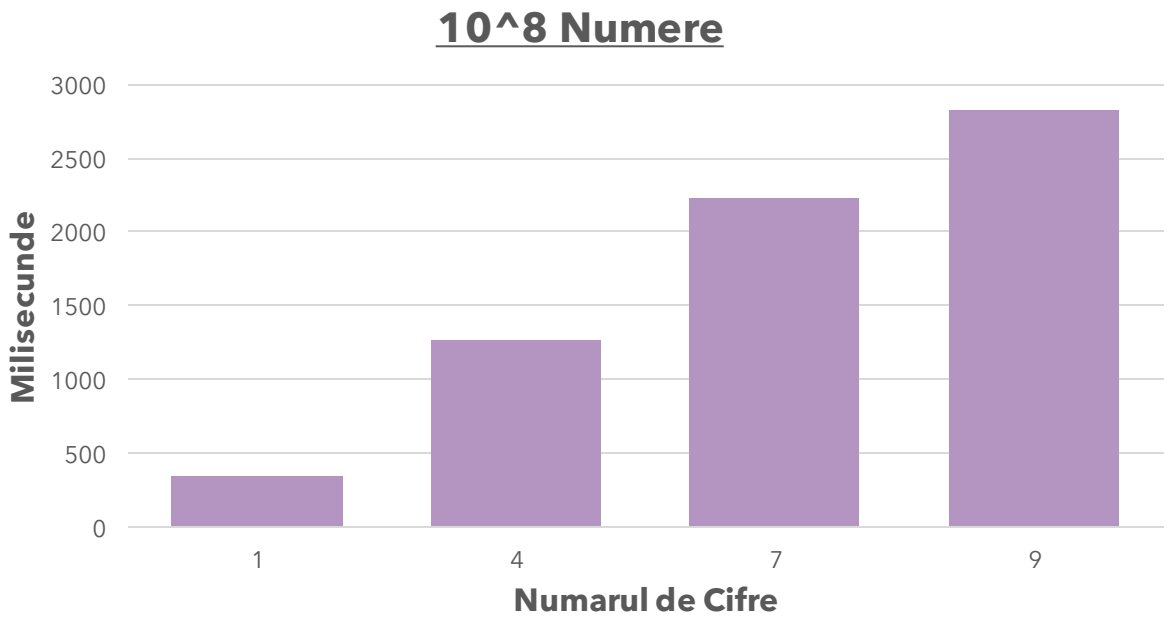
Algoritm Radix Sort

```
void radixsort(vector<long long> &v, long long radixBase, long long nrDigits){
      vector<vector<long long>> B(radixBase);
      long long x = 1, X = 9;
      for (long long i = 0; i < nrDigits - 1; ++i) {
          X = X * 10 + 9;
      }
      while(x < X) {
          B.clear();
          B.resize(radixBase);
          for (long long j = 0; j < v.size(); ++j) {
              B[(v[j] / x) % radixBase].push_back(v[j]);
          }
          v.clear();
          for (long long i = 0; i < radixBase; ++i) {
              for (long long j = 0; j < B[i].size(); j++) {
                  v.push_back(B[i][j]);
              }
          }
          x = x * radixBase;
      }
  }
```

Radix Sort Timp







Merge Sort

Sortarea prin interclasare, sau **Mergesort** este o metodă eficientă de sortare a elementelor unui tablou, bazată pe următoarea idee: dacă prima jumătate a tabloului are elementele sortate și a doua jumătate are de asemenea elementele sortate, prin interclasare se va obține tabloul sortat.

Sortarea prin interclasare este un exemplu tipic de algoritm divide et impera: se sortează o secvență delimitată de indicii **st** și **dr**:

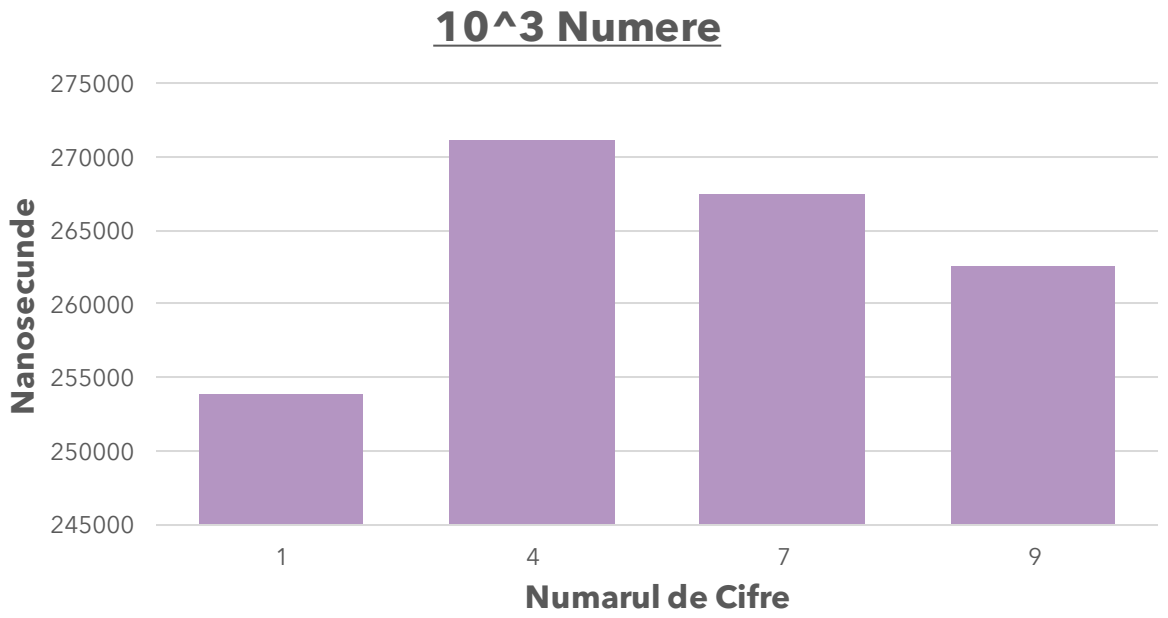
- dacă **st <= dr**, problema este elementară, secvența fiind deja sortată
- dacă **st < dr**:
- se împarte problema în subprobleme, identificând mijlocul secvenței, **m = (st + dr) / 2**;
- se rezolvă subproblemele, sortând secvența delimitată de **st** și **m**, respectiv secvența delimitată de **m+1** și **dr** – apeluri recursive;
- se combină soluțiile, interclasând cele două secvențe; în acest fel, secvența delimitată de **st** și **dr** este sortată.

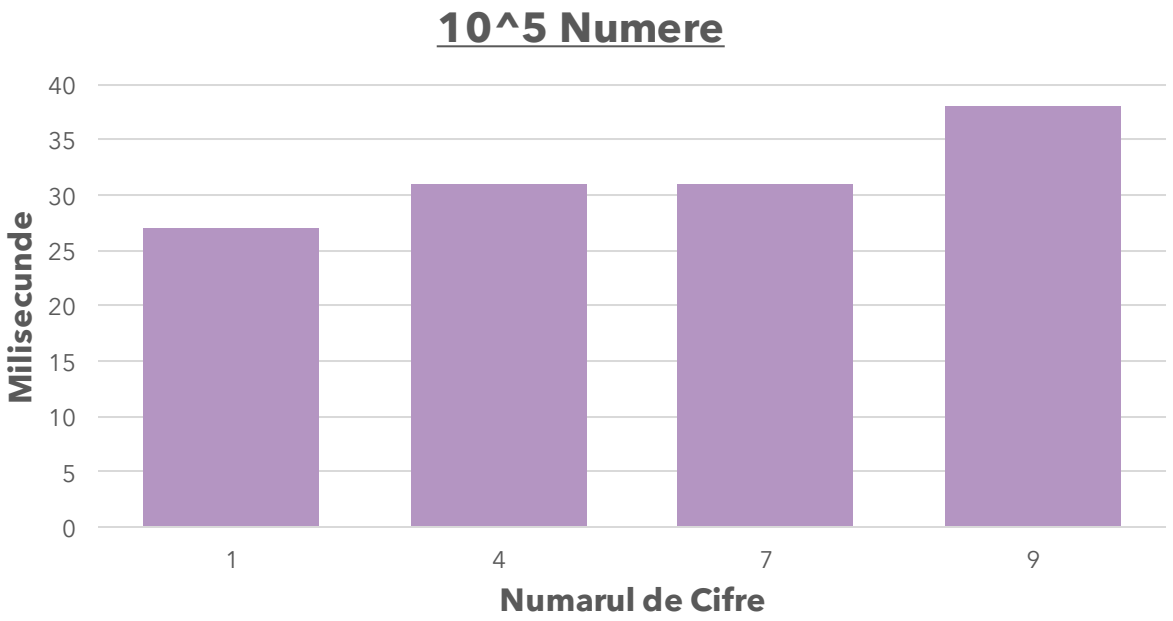
Algoritm Merge Sort

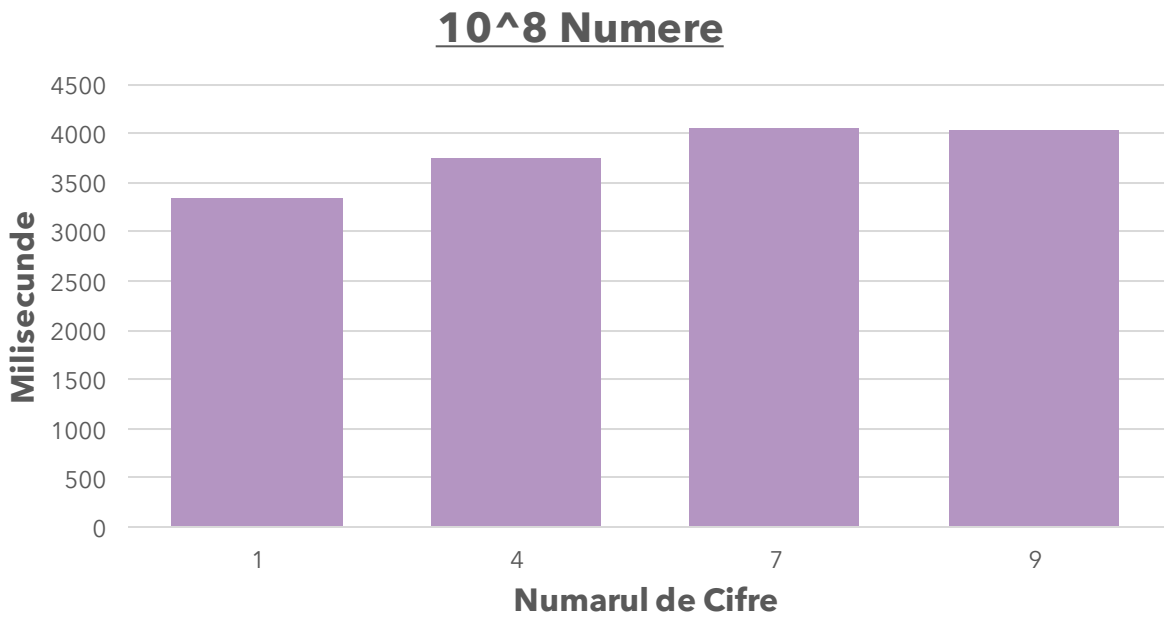
```
void merge(vector<long long>& v, long long start, long
  long mid, long long end) {
      long long i = start, j = mid + 1, k = 0;
      vector<long long> temp(end - start + 1);
      while (i <= mid && j <= end) {
          if (v[i] <= v[j]) {
              temp[k++] = v[i++];
          } else {
              temp[k++] = v[j++];
          }
      }
      while (i <= mid) {
          temp[k++] = v[i++];
      }
      while (j <= end) {
          temp[k++] = v[j++];
      }
      for (i = start, k = 0; i <= end; ++i, ++k) {
          v[i] = temp[k];
      }
  }
```

```
void mergesort(vector<long long>&
v, long long start, long long end) {
    if (start < end) {
        long long mid = (start + end) / 2;
        mergesort(v, start, mid);
        mergesort(v, mid + 1, end);
        merge(v, start, mid, end);
    }
}
```

Merge Sort Timp







Shell Sort

Metoda lui Shell, sau sortarea Shell în structura de date, este un algoritm eficient de sortare de comparație la loc. Este numit după Donald Shell când a propus ideea inițială în 1959. Sortarea Shell este o extensie generalizată a algoritmului de sortare prin inserție.

Ideea fundamentală a acestui algoritm de sortare este de a grupa elementele care sunt îndepărtate și de a le sorta în consecință. Apoi reduceți treptat distanța dintre ele. Sortarea Shell depășește timpul mediu de caz complexsortarea inserției prin compararea și schimbul de elemente aflate la distanță.

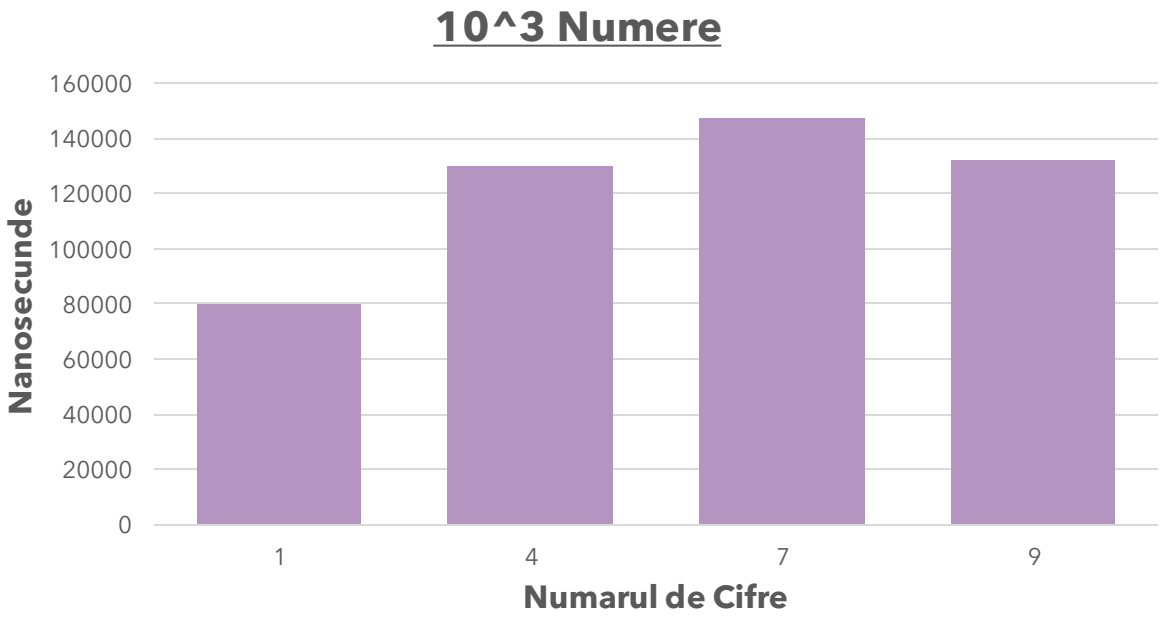
Acest interval, cunoscut sub numele de interval, este redus în funcție de unele secvențe de decalaj optime. Timpul de rulare a sortării shell depinde, de asemenea, de aceste secvențe. Există mai multe secvențe de decalaje, cum ar fi secvența originală a lui Shell, formula lui Knuth, incrementele lui Hibbard etc. Secvența de decalaj inițială a lui Shell este - `n/2, n/4, n/8, ……..1`

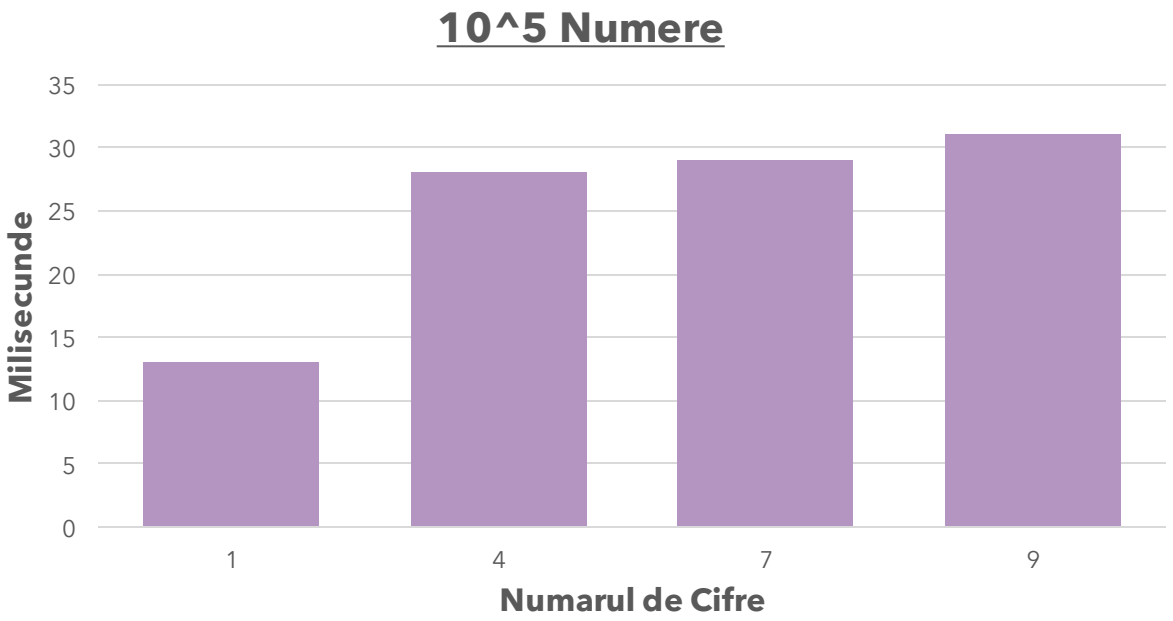
Algoritm Shell Sort

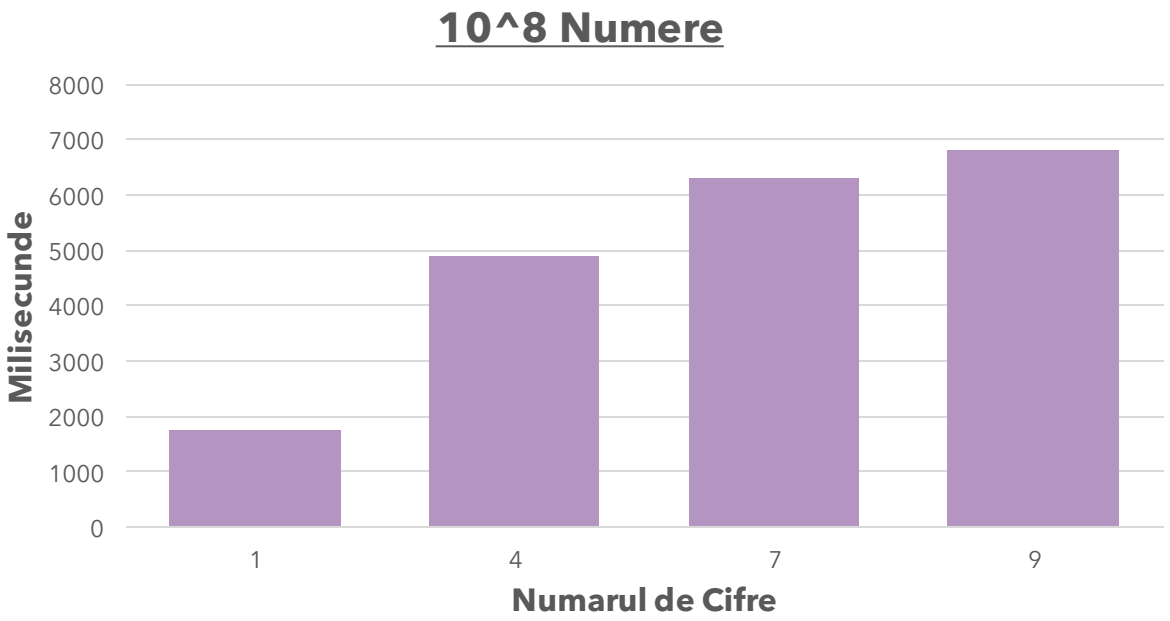
```
void shellSort(vector<long long>& v) {
      long long n = v.size();

      // Gap sequence
      for (long long gap = n/2; gap > 0; gap /= 2) {
          // Do an insertion sort for each gap-sized subarray
          for (long long i = gap; i < n; ++i) {
              long long temp = v[i];
              long long j;
              for (j = i; j >= gap && v[j - gap] > temp; j -= gap) {
                  v[j] = v[j - gap];
              }
              v[j] = temp;
          }
      }
  }
```

Shell Sort Timp







Count Sort

Count sort, cunoscut și sub numele de counting sort, este un algoritm de sortare stabil și non-comparativ care este eficient pentru sortarea unui număr mare de elemente întregi într-un interval cunoscut. Principiul de bază al count sort este de a număra frecvența fiecărui element din lista de intrare și apoi de a utiliza aceste informații pentru a plasa elementele în pozițiile corecte în lista sortată.

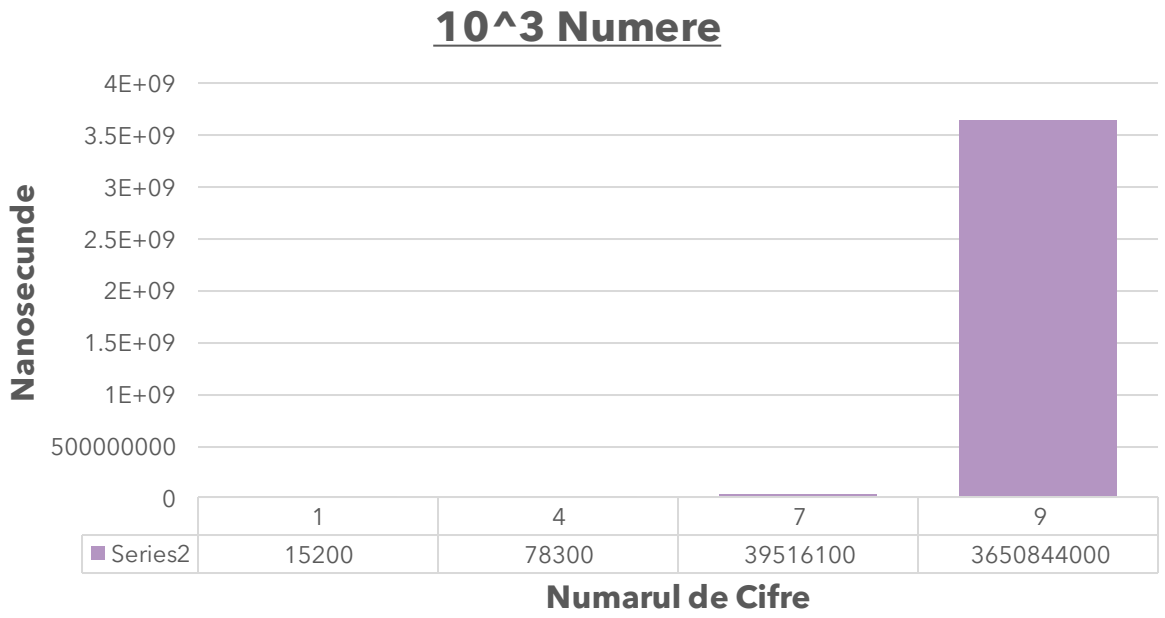
Pașii algoritmului Count Sort:

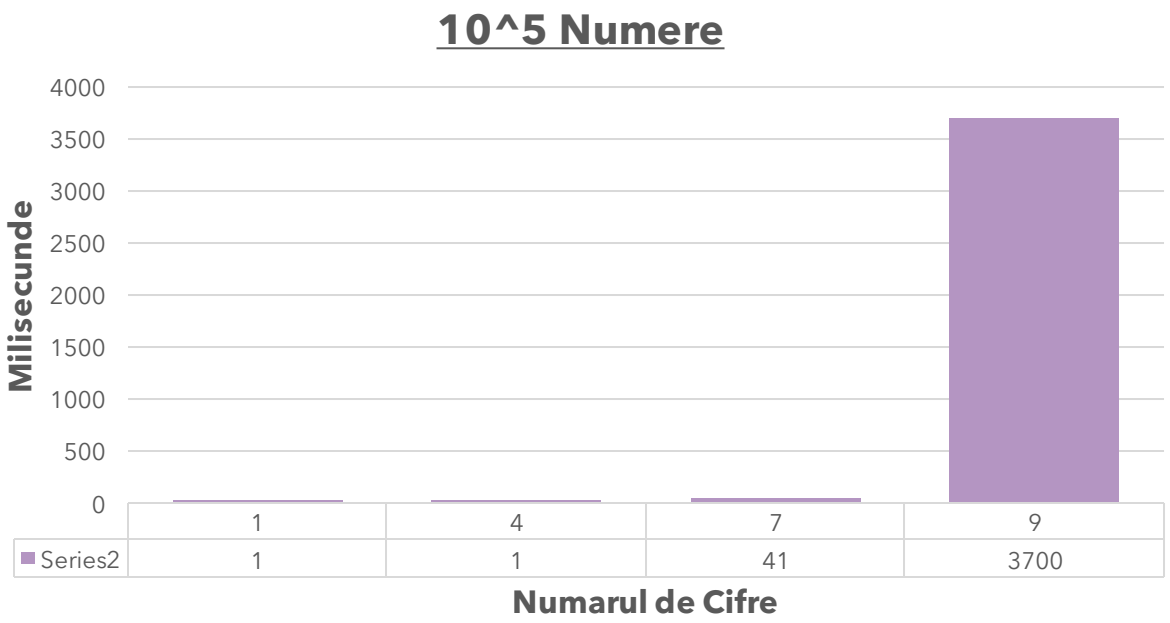
- Se găsesc valorile minime și maxime din lista de intrare pentru a stabili intervalul de valori.
- Se creează un vector de frecvențe (count) de dimensiune egală cu intervalul determinat, inițializat cu zero.
- Se parcurge lista de intrare și se numără aparițiile fiecărui element, incrementând valorile corespunzătoare din vectorul de frecvențe.
- Se transformă vectorul de frecvențe într-un vector de poziții cumulate, astfel încât fiecare poziție să conțină suma frecvențelor anterioare. Acest lucru ajută la determinarea pozițiilor finale ale elementelor în lista sortată.
- Se parcurge lista de intrare de la sfârșit la început și se plasează fiecare element în poziția corectă în lista sortată, utilizând vectorul de poziții cumulate. După plasare, se decrementează valoarea corespunzătoare din vectorul de poziții cumulate.
- Se copiază elementele din lista sortată înapoi în lista originală, dacă este necesar.

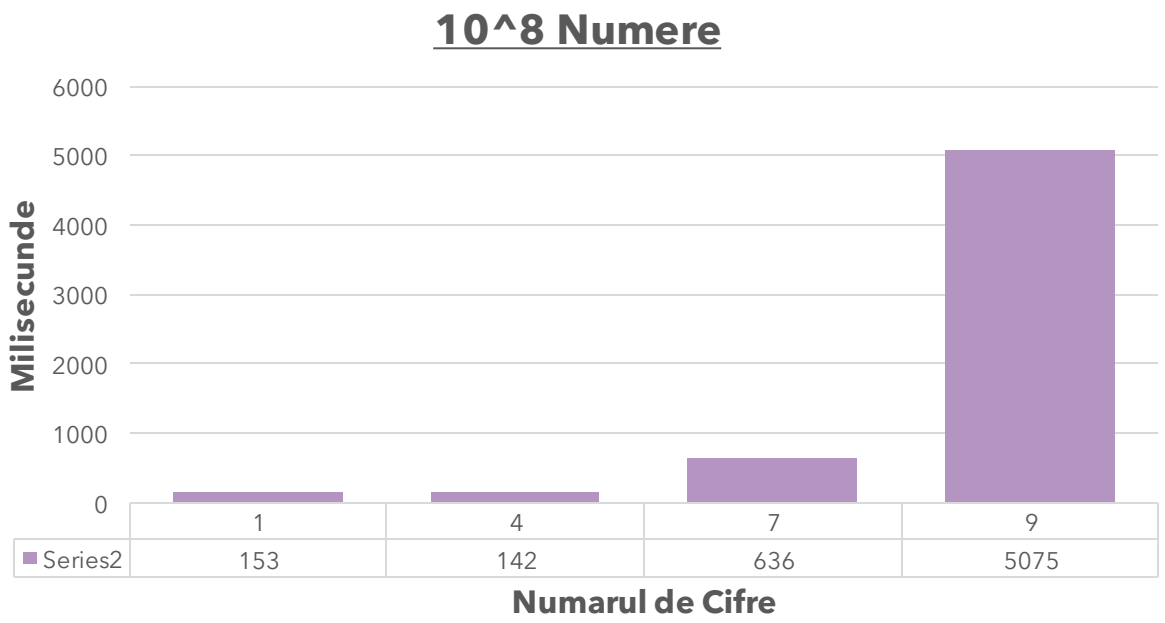
Algoritm Count Sort

```
void countSort(vector<long long> &v, long long nrDigits){
      long long maxx = 1;
      for (long long i = 0; i < nrDigits; ++i) {
          maxx = maxx * 10;
      }
      vector<long long>counter(maxx);
      for (long long i = 0; i < v.size(); ++i) {
          counter[v[i]]++;
      }
      v.clear();
      for (long long i = 0; i < maxx; ++i) {
          for (long long j = 0; j < counter[i]; ++j) {
              v.push_back(i);
          }
      }
  }
```

Count Sort Timp







Quick Sort

Quick sort este un algoritm de sortare eficient și foarte folosit, care utilizează strategia "divide et impera". Alegerea pivotului este un aspect esențial al algoritmului și, în această variantă, pivotul este întotdeauna ultima valoare din sublista curentă.

Pașii algoritmului Quick Sort (cu pivotul ultima valoare):

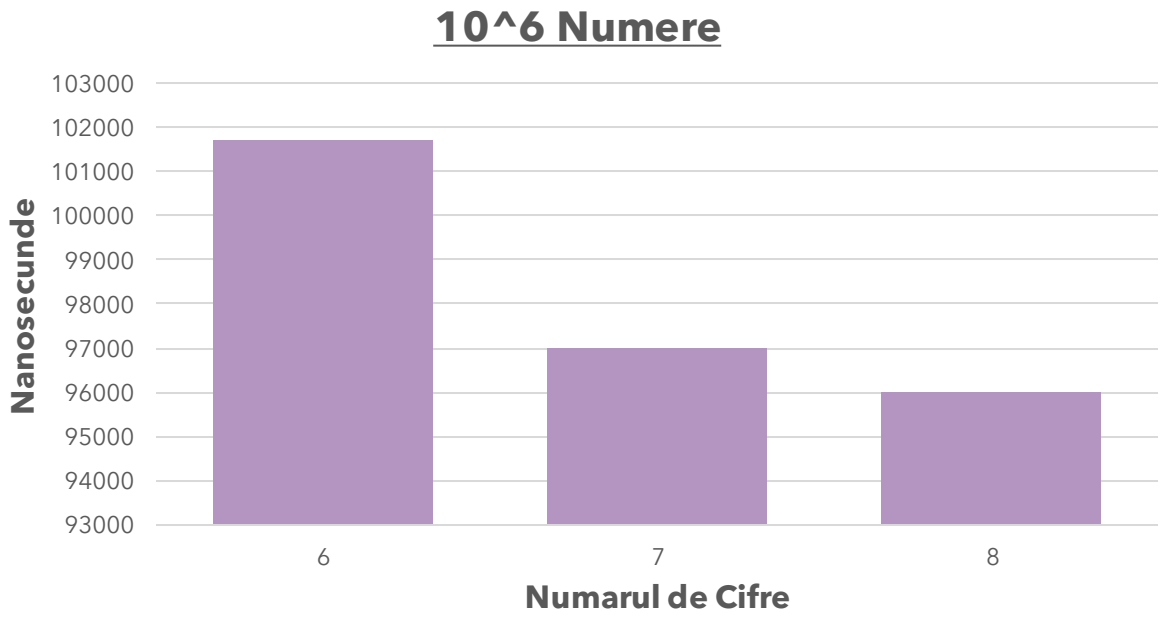
- Se alege pivotul ca fiind ultima valoare din sublista curentă.
- Se mută toate elementele mai mici sau egale cu pivotul în stânga pivotului și toate elementele mai mari în dreapta lui.
- După acest pas, pivotul se află în poziția sa corectă în lista sortată.
- Se aplică algoritmul Quick Sort recursiv pe sublista din stânga pivotului și pe sublista din dreapta pivotului.

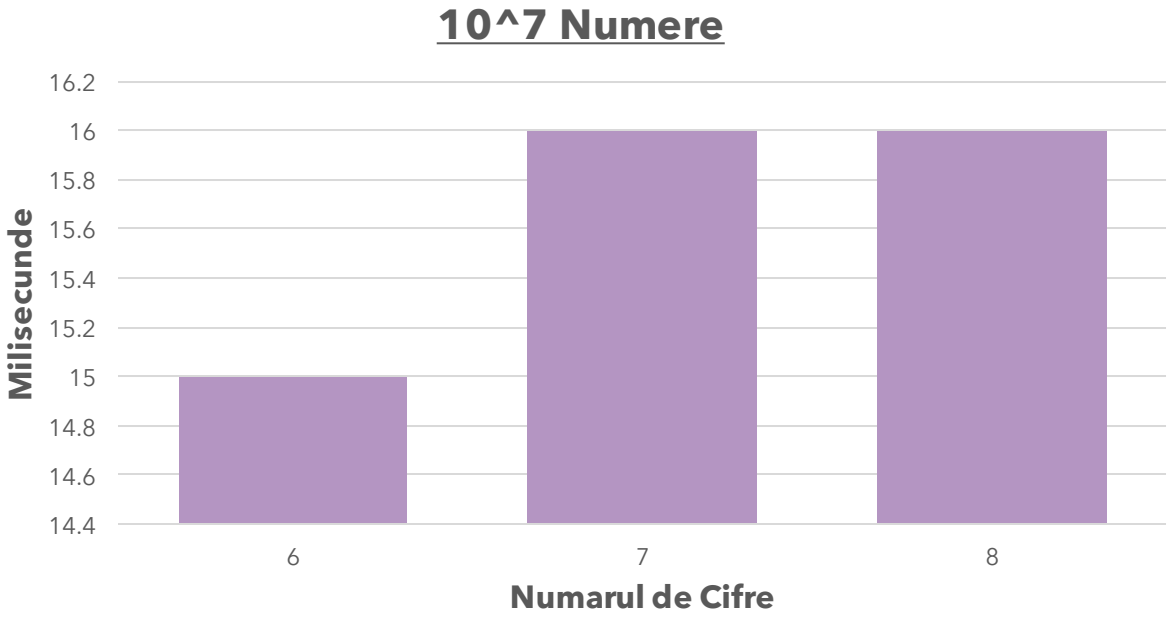
Algoritm Quick Sort

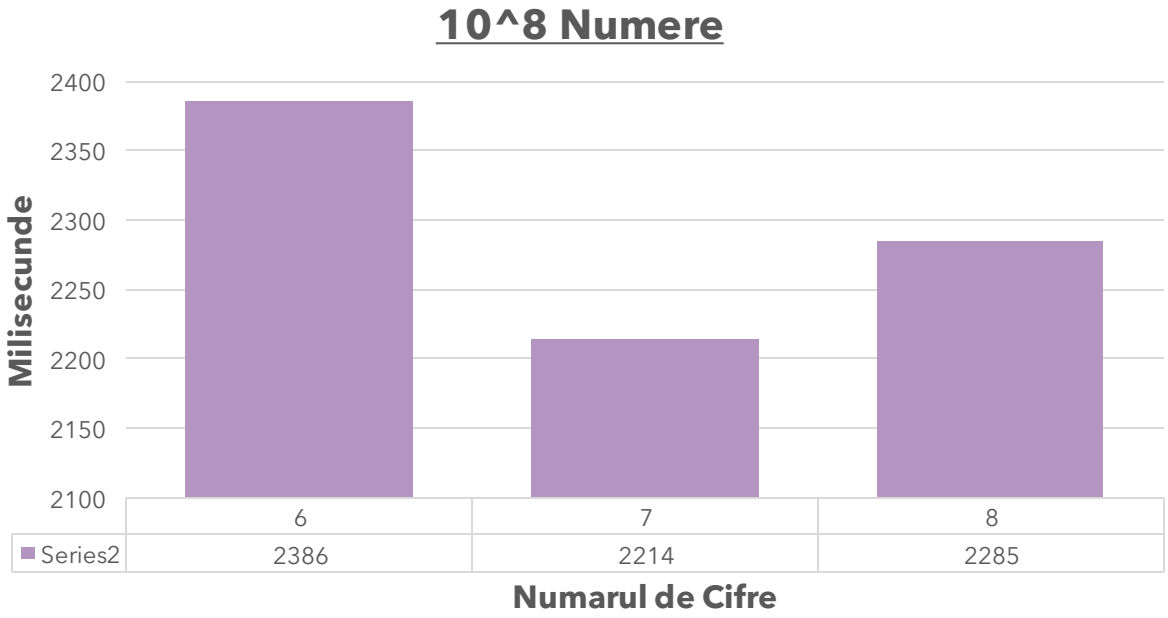
```
long long partition(vector<long long>& v,
  long long start, long long end) {
      long long pivot = v[end];
      long long i = start - 1;
      for (long long j = start; j < end;
  j++) {
          if (v[j] <= pivot) {
              i++;
              swap(v[i], v[j]);
          }
      }
      swap(v[i + 1], v[end]);
      return i + 1;
  }
```

```
void quicksort(vector<long long>& v,
  long long start, long long end) {
      if (start < end) {
          long long pivot = partition(v,
  start, end);
          quicksort(v, start, pivot -
  1);
          quicksort(v, pivot + 1, end);
      }
  }
```

Quick Sort Timp







Bucket Sort

Bucket sort (sau bin sort) este un algoritm de sortare distribuită care este eficient pentru sortarea elementelor distribuite uniform într-un anumit interval. Principiul de bază al bucket sort este de a împărți elementele într-un număr de "buckets" (găleți) și apoi de a sorta individual fiecare bucket folosind un alt algoritm de sortare (de obicei, sortarea prin inserție).

Pașii algoritmului Bucket Sort:

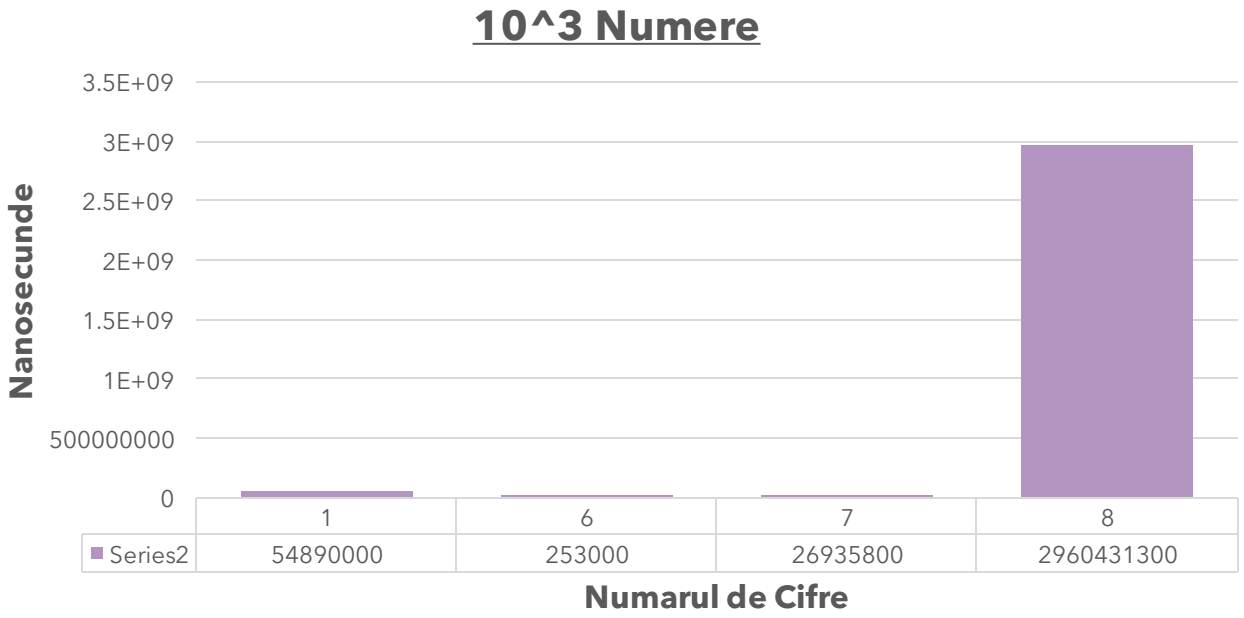
- Se creează un număr fix de bucket-uri.
- Fiecare bucket va conține o sublistă inițializată ca goală.
- Se parcurge lista de intrare și se distribuie fiecare element în bucket-ul corespunzător.
- Alegerea bucket-ului pentru un element se face în funcție de o funcție de hashing sau de o mapare simplă bazată pe valoarea elementului.
- Fiecare bucket se sortează folosind un algoritm de sortare adecvat (de exemplu, sortarea prin inserție).
- Se concatenează toate bucket-urile sortate pentru a obține lista finală sortată.

Algoritm Bucket Sort

```
void bucketsort(vector<long long>& v, long long nrDigits, long long nrNumbers){
      vector<vector<long long>> buckets(pow(10, nrDigits - 1));
      for (long long i = 0; i < nrNumbers; ++i) {
          buckets[v[i] / 10].push_back(v[i]);
          if (buckets[v[i] / 10].size() > 1){
              sort(buckets[v[i] / 10].begin(), buckets[v[i] / 10].end());
          }
      }
      v.clear();
      for (int i = 0; i < buckets.size(); ++i) {
          for (int j = 0; j < buckets[i].size(); ++j) {
              v.push_back(buckets[i][j]);
          }
      }
  }
```

Bucket Sort Timp

Testele pe bucket sort nu au rulat cu succes, iar chiar daca probabil nu este un rezultat real, in imaginea de mai jos am pus un IA sa presupuna cum ar arata.



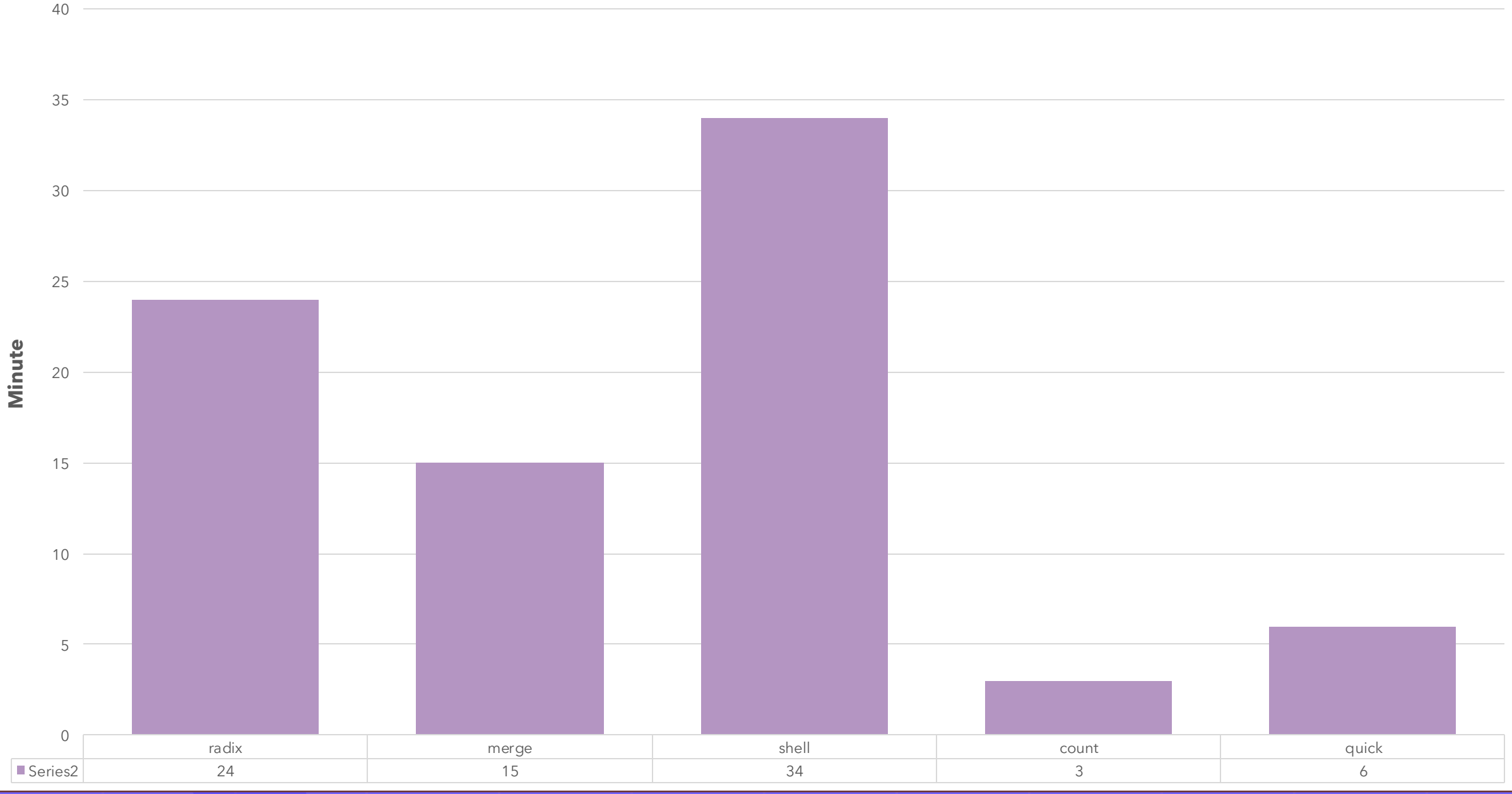
For 100,000 numbers:

1. **1 digit**: Approximately 8.96 seconds
2. **4 digits**: Approximately 42.018 seconds
3. **7 digits**: Approximately 4.316 seconds
4. **9 digits**: Approximately 491.36 seconds

For 10,000,000 numbers:

1. **1 digit**: Approximately 1,242 seconds
2. **4 digits**: Approximately 125.1 seconds
3. **7 digits**: Approximately 598 seconds
4. **9 digits**: Approximately 68,080 seconds ≈ 19 ore

1 miliard de numere cu maxim 9 cifre



Multumesc pentru atentie!

Bibliografie:
https://www.guru99.com
https://www.pbinfo.ro
https://chatgpt.com/